Klaros-Testmanagement
User Manual

Version 4.3.11

Publication date December 16 2015

Copyright © 2009-2015 verit Informationssysteme GmbH

Klaros-Testmanagement User Manual

by Selcuk Akgünlü, Caroline Albuquerque, Sabrina Gidley, Fabian Klaffke, Claudia Könnecke, Klaus Mandola, Patrick Reilly, Tobias Schmitt, Torsten Stolpmann, and Heiner Westphal

Version 4.3.11

Publication date December 16 2015
Copyright © 2009-2015 verit Informationssysteme GmbH

Abstract

This document serves as the reference documentation for the Klaros-Testmanagement application. It gives a detailed description of the user interface and the provided functionality.

Legal Notice. Copyright 2009-2015 verit Informationssysteme GmbH, Europaallee 10, 67657 Kaiserslautern, Germany. All rights reserved. This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or documentation may be reproduced in any form by any means without prior written authorization of verit Informationssysteme GmbH and its licensors, if any.

Trademarks

Oracle™, Java™ and Solaris™ are trademarks of Oracle and/or their affiliates.

Microsoft®, Excel®, SQLServer®, and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

JIRA® is a registered trademark of Atlassian Pty Ltd.

Other names may be trademarks of their respective owners.

Third party software and licenses

This product contains software covered by the following licenses:

OTN License Agreement (OTN) . This application contains the Oracle JDBC Driver. Please read http://www.oracle.com/technology/index.html for more details on JDBC Driver License agreement. JDBC Driver and all associated intellectual property rights are retained by Oracle Corporation and/or its licensors. To use JDBC Driver included this application, you need to agree with Oracle Technology Network Development and Distribution License Terms. If you don't, you can't use this application.

REDISTRIBUTION LICENSE FOR MICROSOFT JDBC DRIVER 4.0 FOR SQL SERVER (OTN) . This application contains the Microsoft SQLServer JDBC Driver 4.0. Please read http://download.microsoft.com/download/0/2/A/02AAE597-3865-456C-AE7F-613F99F850A8/license.txt for more details on the Microsoft SQLServer JDBC Driver License agreement. Microsoft SQLServer JDBC Driver and all associated intellectual property rights are retained by Microsoft and/or its licensors. To use the Microsoft SQLServer JDBC Driver included this application, you need to agree with MICROSOFT JDBC DRIVER 4.0 FOR SQL SERVER MICROSOFT SOFTWARE LICENSE TERMS. If you don't, you can't use this application.

Lesser GNU Public License (LGPL 2.1) .

- JasperReports

This product uses software available under the Apache Software License (ASL 2.0) .

This product uses icons from the Tango Desktop Project (http://tango.freedesktop.org/) which are released to the Public Domain. We thank the authors for their worthwhile efforts. .

This product uses icons from the Fugue Icon Set (http://www.pinvoke.com/) which are available under a Creative Commons Attribution 3.0 license. .

This product uses icons from Patricia Clausnitzer's pc.de Icon Sets (http://pc.de/icons/) which are available under a Creative Commons Attribution 3.0 license. .

Table of Contents

1. Key Features 1
    1.1. Klaros-Testmanagement Community Edition Features 1
    1.2. Klaros-Testmanagement Enterprise Edition Features 2
    1.3. Klaros-Testmanagement Mobile Edition Features 3
2. Introduction 5
    2.1. What is Klaros-Testmanagement 5
    2.2. Test Cases - The Basis of Testing 5
    2.3. Test Suites - Organizing Test Cases 6
    2.4. Test Runs - Executed Tests with their Results 7
    2.5. Test Environment and System under Test - Influences of Test Results 7
    2.6. Iterations - Subdividing your project into phases 7
    2.7. Requirements – Meeting Quality Standards 8
    2.8. Jobs - Planning the test process 8
    2.9. User Roles - Every User has his Rights 9
    2.10. Overview of Artifacts 10
    2.11. Issue Management Systems 10
3. Installation 11
    3.1. License Model 11
        3.1.1. License Model (Users) 11
        3.1.2. License Model (Mobile Devices) 11
    3.2. Prerequisites 12
        3.2.1. Client Prerequisites 12
        3.2.2. Server Prerequisites 12
    3.3. Installation Process 19
        3.3.1. Step 1: Welcome 19
        3.3.2. Step 2: Information 20
        3.3.3. Step 3: Licensing Agreements 21
        3.3.4. Step 4: Target Path 22
        3.3.5. Step 5: Select Installation Packages 23
        3.3.6. Step 6: User Data 24
        3.3.7. Step 7: Installation 24
        3.3.8. Step 8: Perform External Processes 26
        3.3.9. Step 9: Setup Shortcuts 26
        3.3.10. Step 10: Installation Finished 27
    3.4. Console based installation 28
        3.4.1. Step 1: Welcome 28
        3.4.2. Step 2: Information 29
        3.4.3. Step 3: Licensing Agreements 29
        3.4.4. Step 4: Target Path 30
        3.4.5. Step 5: Select Installation Packages 30
        3.4.6. Step 6: User Data 30
        3.4.7. Step 7: Installation 31
        3.4.8. Step 8: Perform External Processes 31
        3.4.9. Step 9: Setup Shortcuts 31
        3.4.10. Step 10: Installation Finished 31
    3.5. Automated Installation Script 32
    3.6. Update Process 33

3.6.1. Incompatible Installation Detected 33
3.6.2. Manual Changes 34
3.6.3. Database Migration 35
3.7. Important File Locations 36
3.7.1. Log Files 36
3.7.2. The Hibernate Properties File 37
3.7.3. Language Files 37
3.7.4. The Quotes File 37
3.7.5. The Derby Database 37
3.7.6. The Attachment Repository 37
3.8. Running Klaros-Testmanagement 37
3.9. Changing the Default Database 37
3.10. Installing Klaros-Testmanagement as a Service 39
3.10.1. Installing Klaros-Testmanagement as a Linux service 39
3.10.2. Installing Klaros-Testmanagement as a Windows service 39
3.11. Monitoring Klaros-Testmanagement 41
3.12. Configuring External Issue Management Systems 42
3.12.1. JIRA Configuration 42
3.12.2. Redmine Configuration 42
3.12.3. Trac Server Configuration 44
3.13. SSL Support 46
3.14. Uninstall 46
4. Customization 47
4.1. Languages 47
4.1.1. Defining Language Files 47
4.2. Custom Quotes 48
5. Functional Overview 49
5.1. Login 49
5.2. Main Functions 51
5.2.1. The Help Menu 51
5.2.2. The User Menu 52
5.2.3. Search / Quick-Select 52
5.2.4. The Log Panel 54
5.2.5. Filtering and Sorting 54
5.2.6. Print Pages 56
5.2.7. Categorization 57
5.2.8. Conflict Handling 59
5.2.9. Deleting Objects 60
5.2.10. Bulk Actions 60
5.2.11. User Defined Properties 61
5.2.12. Attachments 63
5.2.13. Revisions 64
5.2.14. Related Test Runs and Results 64
5.2.15. Change History 65
5.2.16. Overview Tabs 66
6. Define 68
6.1. Projects 68
6.1.1. Creating a Project 69
6.1.2. Deleting a Project 69

6.1.3. Restoring a Project 69
6.1.4. Purging a Project 69
6.1.5. Editing the Description of a Project 69
6.1.6. Project Details 69
6.2. Iterations 73
6.2.1. Bulk Actions 74
6.2.2. Creating an Iteration 74
6.2.3. Iteration Details 75
6.3. Requirements 79
6.3.1. Bulk Actions 80
6.3.2. Creating a Requirement 80
6.3.3. Requirement Details 81
6.4. Test Environments 83
6.4.1. Bulk Actions 84
6.4.2. Creating a Test Environment 85
6.4.3. Test Environment Details 85
6.5. Systems under Test 87
6.5.1. Creating a System under Test 88
6.5.2. Bulk Actions 88
6.5.3. System under Test Details 88
6.6. Test Cases 91
6.6.1. Bulk Actions 92
6.6.2. Creating Test Cases 93
6.6.3. Test Case Details 93
6.7. Test Suites 99
6.7.1. Bulk Actions 99
6.7.2. Create Test Suites 100
6.7.3. Test Suite Details 100
7. Plan 104
7.1. Maintain Jobs 104
7.1.1. Bulk Actions 104
7.1.2. Creating Jobs 105
7.1.3. Arranging Jobs 105
7.1.4. Executing a Job 105
7.1.5. Job Details 106
7.1.6. Jobs from Test Cases 111
7.1.7. Jobs from Test Suites 111
7.1.8. Jobs by User 112
8. Execute 114
8.1. My Jobs 114
8.2. Run Test Case 114
8.2.1. Executing a Single Test Case 114
8.3. Run Test Suite 120
8.3.1. Executing a Test Suite 120
8.4. Continue Test Run 123
8.4.1. Continuing a Suspended Test Run 124
8.5. Import Test Results 124
8.6. Creating Issues 127
8.7. Creating Review Jobs 132

9. Evaluate 134
9.1. Dashboard 134
9.1.1. The Overview Reports 134
9.1.2. The Dashboard Reports 137
9.2. Report Templates 142
9.3. Test Run History Report 143
9.4. Test Runs 144
9.4.1. The Test Run Report 146
9.5. Test Case Results 147
9.5.1. Single Test Case Results 148
9.6. Test Suite Results 152
9.6.1. Single Test Suite Results 153
9.7. Issues 154
9.7.1. Issues by Test Case 155
9.7.2. Issue Details (Creating a new Issue) 155
9.7.3. Link Issues 158
9.8. Bookmarking Results 159
10. Configure 160
10.1. Overview 160
10.2. Report Templates 160
10.2.1. Report Details 161
10.3. Users 162
10.3.1. Creating a User 163
10.3.2. Deleting a User 164
10.3.3. User Details 164
10.4. Mobile Devices 166
10.4.1. Mobile Device Details 168
10.5. System 169
10.5.1. Miscellaneous Settings 169
10.5.2. Notification Settings 170
10.5.3. Interface Settings 172
10.5.4. Languages 174
10.6. Integration 174
10.6.1. Issue Management 174
10.6.2. E-Mail 178
10.6.3. Network 180
10.6.4. LDAP 180
10.7. Backup 183
10.7.1. Export 183
10.7.2. Import 184
11. Custom Reports 186
11.1. Create A New Report Template 187
11.1.1. Supported Parameter Types 189
11.1.2. Dealing with Parameters 189
11.2. Applying a Report Template 190
11.3. Example Report 191
11.3.1. Creating the Groovy Script 191
11.3.2. Creating a SeamPDF Report Template 192
11.3.3. Creating a Chart 192

11.3.4. Including Images 194
11.3.5. Creating a SeamExcel Report Template 194
12. Import/Export 196
12.1. Importing Test Cases from Excel 196
12.1.1. Import Format 196
12.1.2. Prerequisites 198
12.1.3. Execution 198
12.2. Importing Test Cases from XML 199
12.2.1. Prerequisites 199
12.2.2. Execution 199
12.3. Importing Requirements from Excel 200
12.3.1. Import Format 200
12.3.2. Prerequisites 201
12.3.3. Execution 202
12.4. Importing and Synchronizing Requirements from XML 202
12.4.1. Prerequisites 202
12.4.2. Importing 203
12.4.3. Synchronizing 203
12.5. Importing Test Results 205
12.5.1. Prerequisites 205
12.5.2. Execution 205
12.6. Exporting Table Content to Excel 208
12.7. Backup/Recovery 208
Glossary 209
A. Role Permission Overview 214
B. Model API Reference 216
B.1. API-Reference 216
B.1.1. de.verit.klaros.core.model 216
B.2. Scripting API-Reference 322
B.2.1. de.verit.klaros.scripting 322
B.2.2. de.verit.klaros.scripting.context 326
B.2.3. de.verit.klaros.scripting.custom 330
B.2.4. de.verit.klaros.scripting.model 336
B.2.5. Constant field values 342
C. Test Case Import File Specification 343
C.1. <step> 345
C.2. <steps> 345
C.3. <attribute> 345
C.4. <attributes> 345
C.5. <container> 346
C.6. <testcases> 346
C.7. <testcase> 346
C.8. <triggers> 347
C.9. <testsuite> 347
C.10. <externalId> 347
C.11. <externalRevision> 347
C.12. <trigger> 347
C.13. <areatopic> 348
C.14. <depends> 348

C.15. <description> 348
C.16. <docbase> 348
C.17. <evaluation> 348
C.18. <execution> 348
C.19. <expectedResult> 348
C.20. <level> 348
C.21. <method> 349
C.22. <note> 349
C.23. <postcondition> 349
C.24. <precondition> 349
C.25. <revision> 349
C.26. <priority> 349
C.27. <shortname> 349
C.28. <state> 349
C.29. <team> 349
C.30. <traceability> 350
C.31. <variety> 350
C.32. <name> 350
C.33. <value> 350
D. Requirement Import File Specification 351
D.1. <attribute> 352
D.2. <attributes> 352
D.3. <container> 352
D.4. <requirements> 352
D.5. <requirement> 352
D.6. <triggers> 353
D.7. <externalId> 353
D.8. <externalRevision> 353
D.9. <trigger> 353
D.10. <description> 354
D.11. <priority> 354
D.12. <revision> 354
D.13. <shortname> 354
D.14. <summary> 354
D.15. <name> 354
D.16. <value> 354
E. Reporting Resources 355
E.1. Context Variables 355
E.2. KlarosScript Interface 355
E.3. Example report template 355
Index 359

List of Figures

2.1. Test Case Structure 5
2.2. Test Case Step 5
2.3. A Test Suite 6
2.4. Test Run with Test Results 7
2.5. Jobs 9
2.6. Artifacts Overview 10
3.1. Welcome Screen 20
3.2. Information Screen 21
3.3. Licensing Agreement Screen 22
3.4. Target Path Screen 23
3.5. Select Installation Packages Screen 23
3.6. User Data Screen 24
3.7. Installation in Progress Screen 25
3.8. Installation Finished Screen 25
3.9. Perform External Processes Screen 26
3.10. Setup Shortcuts Screen 27
3.11. Installation Finished Screen 28
3.12. Incompatible Installation detected 34
3.13. The Uninstaller 34
3.14. Database Migration 35
3.15. Additional Tomcat memory settings 41
3.16. Redmine Plugin Section 43
3.17. Redmine Authentication Section 44
3.18. Trac Plugin Section 45
3.19. Trac Permissions Section 45
5.1. Login Screen 49
5.2. Maintain Projects Page 50
5.3. The Help Menu 51
5.4. The Help Menu Button 52
5.5. The Help Menu 52
5.6. The Search Field 52
5.7. The Search Results Page 53
5.8. The Status Bar Search Report 53
5.9. The Log Panel 54
5.10. The Log Panel Icon 54
5.11. The Filter and Sort Panel 54
5.12. The Test Suite Print Page 57
5.13. The Categorization Edit View 58
5.14. The Categorization Tree View 58
5.15. The Categorization Menu View 59
5.16. The Conflict Resolution Dialog 59
5.17. The Delete Objects Popup 60
5.18. The Bulk Actions on the Test Case List Page 61
5.19. Bulk Editing Objects 61
5.20. The User Defined Tab 62
5.21. Editing an Enumeration Property 63
5.22. The Test Suite User Defined Tab 63

5.23. The Assign Attachments Screen 64
5.24. The Test Suite Revisions Tab 64
5.25. The Results Tab 65
5.26. The Test Case Changes Tab 66
5.27. The Test Case Overview Tab 67
6.1. The Projects Screen 68
6.2. The Project Properties Page 70
6.3. Copying Objects to the Current Project 71
6.4. Copying Objects from another Project 71
6.5. The Project Access tab 72
6.6. The Project Access Popup 73
6.7. The Iteration and Project Selection Toolbar 73
6.8. The Iterations Screen 74
6.9. The Iteration Overview Tab 75
6.10. The Iteration Success Rate 75
6.11. The Iteration Health Matrix 76
6.12. The Iteration Details Screen 78
6.13. The Requirements Screen 80
6.14. The Requirement Overview Tab 81
6.15. The Requirement Details Screen 83
6.16. The Test Environments Screen 84
6.17. The Test Environment Overview Tab 85
6.18. The Systems under Test Screen 87
6.19. The System under Test Overview Tab 89
6.20. The Test Case Issues Screen 91
6.21. The Test Cases Screen 92
6.22. The Test Case Overview Tab 93
6.23. The Edit Test Cases Screen 96
6.24. The Edit Test Steps Screen 97
6.25. The Test Case Issues Screen 98
6.26. The Test Case Jobs Screen 98
6.27. The Test Suites Screen 99
6.28. The Test Suite Details Screen 101
6.29. The Test Suite Overview Tab 101
6.30. The Test Suite Jobs Screen 102
7.1. The Maintain Jobs Screen 104
7.2. The Job Overview Tab 106
7.3. The Job States 108
7.4. The Job Details Screen 109
7.5. The Add a Comment Dialog 110
7.6. The Log Work Dialog 111
7.7. The Jobs from Test Cases Screen 111
7.8. The Jobs from Test Suites Screen 112
7.9. The Jobs by User Screen 112
7.10. The Jobs By User Screen 113
8.1. The My Jobs Screen 114
8.2. The Run Test Case Screen 115
8.3. The Single Test Case Execution Dialog 115
8.4. The Step-by-step Instructions View 116

8.5. The Tabular Step Instructions Screen 117
8.6. The Error or Failure Detected Dialog 119
8.7. The Test Case Results Screen 120
8.8. The Run Test Suite Screen 120
8.9. The Detailed Information about the Test Suite Screen 121
8.10. The Overview Screen 122
8.11. The Test Suite Results Screen 123
8.12. The Continue Test Run Screen 124
8.13. Import Results Page 126
8.14. The Import Test Results Page 127
8.15. Jira Issue Page 128
8.16. Trac Issue Page 129
8.17. Redmine Issue Page 130
8.18. Bugzilla Issue Page 131
8.19. Mantis Issue Page 132
8.20. Creating a Review Job during Execution 133
9.1. The Dashboard Screen 134
9.2. The Test Environment Overview Report Layout 135
9.3. The SUT Overview Report Layout 136
9.4. The Test Suite Overview Report Layout 137
9.5. The Project Overview Report 138
9.6. The Latest Success Rate Report 139
9.7. The Test Activity Report 140
9.8. The Project Health Matrix Report 140
9.9. The Test Progress Report 141
9.10. The Test Progress History Report 142
9.11. User Defined Reports Screens 142
9.12. Generate a parameterized Report 143
9.13. The Test Run History Report 143
9.14. The Test Run History Report Layout 144
9.15. The Test Runs Screen 145
9.16. The Test Run Details Screen 145
9.17. The Test Run Info Column 145
9.18. The Single Test Run Report 146
9.19. The Single Test Run Report (continued) 147
9.20. The Test Case Results Screen 147
9.21. The Test Suite from Test Case Results Pop-up 148
9.22. The Single Test Case Results Screen 148
9.23. The Test Case Results Details Screen 149
9.24. Editing Test Case Step Results 150
9.25. The Test Case Results Print Screen 151
9.26. The Location Tab 152
9.27. The Test Suite Results Screen 152
9.28. The Test Suite Results Screen 153
9.29. The Test Suite Results Screen - Test Results 153
9.30. The Test Suite Results Screen - Print Test Results 154
9.31. The Issues by Test Case Screen 155
9.32. The Create Issue Page 155
9.33. The Issues by Test Case Details Screen 157

9.34. The Test Case Selection Screen 158
9.35. The Link Issues Screen 158
9.36. The Test Case Results Screen 159
10.1. The Overview Page 160
10.2. User-Defined Report Template Page 160
10.3. Generate a parameterized Report 161
10.4. Report Details 162
10.5. The Maintain Users Page 163
10.6. The Save New User Dialog 163
10.7. The Edit User Screen 165
10.8. The Project Roles Tab 165
10.9. The Project Roles Tab 166
10.10. The Mobile Device Connection Log Tab 166
10.11. The Mobile Devices Page 167
10.12. The Mobile Devices Details Page 168
10.13. The Mobile Device Results Tab 168
10.14. The Mobile Device Connection Log Tab 169
10.15. The Miscellaneous Settings Tab 169
10.16. The Notification Schemes Tab 171
10.17. The Notifications Tab 172
10.18. Assigning Projects to a Notification Scheme 172
10.19. The Interface Settings Tab 173
10.20. The Languages Tab 174
10.21. The Issue Management Tab 175
10.22. The Bugzilla Project ID 176
10.23. The JIRA Project ID 177
10.24. The Mantis Project ID 177
10.25. The Redmine Project ID 178
10.26. The E-Mail Tab 179
10.27. The Network Tab 180
10.28. The LDAP Tab 181
10.29. The LDAP Authentication Popup 183
10.30. The Export Tab 184
10.31. The Import Tab 184
11.1. The Report Generation Process 186
11.2. The Report Templates Page 187
11.3. The New Report Templates Page 188
11.4. Specifying Parameters 189
11.5. Apply a Report Template 190
11.6. Enter Parameter 191
11.7. A Pie Chart Example 193
12.1. Test Case Excel Sheet Sample 196
12.2. Requirement Excel Sheet Sample 200
12.3. Export Table Content to Excel 208

List of Tables

3.1. Supported Browsers 12
3.2. Supported Databases 12
3.3. Supported External Issue Management Systems 42
12.1. General Property Coordinates 197
12.2. Test Case Step Coordinates 197
12.3. User Defined Property Coordinates 198
12.4. General Property Coordinates 201
12.5. User Defined Property Coordinates 201
A.1. Role Permission Overview Table - Klaros-Testmanagement Community Edition 214
A.2. Role Permission Overview Table - Klaros-Testmanagement Enterprise Edition 214
A.3. Role Permission Overview Table - Klaros-Testmanagement Mobile Edition 214
B.1. KlarosQueryFactory 342
C.1. Element summary 343
C.2. <step> elements 345
C.3. <steps> elements 345
C.4. <attribute> elements 345
C.5. <attributes> elements 345
C.6. <container> elements 346
C.7. <testcases> elements 346
C.8. <testcase> elements 346
C.9. <triggers> elements 347
D.1. Element summary 351
D.2. <attribute> elements 352
D.3. <attributes> elements 352
D.4. <container> elements 352
D.5. <requirements> elements 352
D.6. <requirement> elements 352
D.7. <triggers> elements 353
E.1. Context Variables 355

List of Examples

3.1. sample auto-install.xml 32
3.2. Additional entries in mysql.cnf to support UTF-8 character sets 39
12.1. Excel Test Case Import via Command Line 198
12.2. Excel Test Case Import via Command Line with authentication 199
12.3. XML Test Case Import via Command Line 199
12.4. XML Test Case Import via Command Line with authentication 200
12.5. Excel Requirement Import via Command Line 202
12.6. Excel Requirement Import via Command Line with authentication 202
12.7. XML Requirement Import via Command Line 203
12.8. XML Requirement Import via Command Line with authentication 203
12.9. XML Requirement Synchronization via Command Line 204
12.10. XML Requirement Import via Command Line with authentication 205
12.11. QF-Test import URL sample 207
12.12. curl command line example 207

Chapter 1. Key Features

1.1. Klaros-Testmanagement Community Edition Features

Key features of Klaros-Testmanagement Community Edition:

Management of Test-related Artifacts	Klaros-Testmanagement allows management of projects, iteration, requirements, test cases, test suites, test environments, systems under test, test runs and test results.
Revisioning	Klaros-Testmanagement supports revisioning of requirements, test cases and test suites. Different versions can be applied to individual systems under test and test environments with full traceability of their results.
Saving Binary Attachments	Klaros-Testmanagement allows to save binary content of any type (text documents, graphics, screenshots etc.) to test cases and their results, iterations, requirements, systems under test, test environments and test suites.
Reports and Statistics	With Klaros-Testmanagement, reports and statistics for test results can easily be assembled. All test related data is stored in a database and can be accessed at any time.
Report File Export	The report files can be exported into various file formats, such as PDF, HTML, CSV and TXT.
Guided Manual Test Execution	Klaros-Testmanagement supports the tester during execution of manual tests. A web-based client leads the tester through the test steps, allows him to attach comments and attachments and automatically records the test process and results.
Continuation of Interrupted Work	Manual execution can be paused whenever and be resumed any later time.
Security	Klaros-Testmanagement supports a fine grained role based user management system connected with a rights and role system. In this way the access and editing rights of single users can be restricted for critical data.
Interoperability with continuous integration servers (Jenkins/Hudson)	A Jenkins/Hudson plug-in enables transfer of results of automated tests on the continuous integrations server to Klaros-Testmanagement.
Integration with Issue Management Systems	Integration with Bugzilla, JIRA, Mantis, Redmine and Trac enable direct documentation of issues found while executing a test.

Import Interfaces for Test Cases	Test case definitions can be imported in well-defined XML and Excel formats.
Import Interface for Test Automation	Test results from test automation software (e.g. JUnit) can be easily imported and added to the manual test results. Supported test automation software includes: *AUnit*, *Boost Test*, *Check*, *CppTest*, *CppUnit*, *ctest*, *CUnit*, *Free Pascal Unit*, *Jubula / GUIDancer*, *JUnit*, *MbUnit*, *MSTest*, *NUnit*, *PHPUnit*, *QF-Test*, *QTestLib*, *Ranorex*, *Selenium*, *TestComplete*, *TestNG*, *TUSAR*, *UnitTest++*, *Valgrind* and *xUnit.net*.
Backup and Restore	Complete test projects can be backed up and restored. The restore interface can also be used to import foreign data from XML files.

1.2. Klaros-Testmanagement Enterprise Edition Features

Additional features of Klaros-Testmanagement Enterprise Edition:

Project based role management	A user can be assigned to a different role in each project, allowing to individually limit rights and access to information.
Agile Test Management (Scrum)	Iterative development processes are supported by using test cycles to coincide with development cycles (sprints).
Requirement Management, Test Coverage	Collection of requirements and linking them to test cases enables Klaros-Testmanagement to track the test coverage of the requirements.
Assigning and Management of Jobs	Jobs to execute test cases and test suites can be generated, assigned to users and tracked easily at all times.
Evaluation of the User Workload	With the help of tables and diagrams, the workload, duration and progress of test activity for each user can be clearly displayed.
Measure Success Rate and Progress	Jobs can be arranged hierarchically. The success rate and progress of all jobs can be seen at any time.
Categorization	Klaros-Testmanagement Enterprise Edition allows multiple categorization of requirements, iterations, systems under test, test environments, test cases and test suites.
Change history	Every change to test data is recorded and displayed in the change history to enable better tracking of changes.
User-Defined Custom Fields	Klaros-Testmanagement Enterprise Edition allows the definition and usage of custom fields for requirements, iter-

	ations, test cases, test suites, test environments and systems under test.
User Definable Reports	Klaros-Testmanagement Enterprise Edition also supports definition and creation of individually designed reports in PDF or Excel format. For this a simple but powerful programming interface is integrated.
Configurable Dashboard	In Klaros-Testmanagement Enterprise Edition, users can organize their dashboard according to their individual requirements.
LDAP Support for User Authentication	Klaros-Testmanagement Enterprise Edition can authenticate users against an external LDAP/Active Directory system which contains the passwords of the users.
Excel Export of Data Tables	Klaros-Testmanagement Enterprise Edition enables the quick export of data tables in excel format in order to undertake further processing on demand.
Print View	All important test data in Klaros-Testmanagement Enterprise Edition can be displayed on printer friendly pages.
Bulk Actions	Klaros-Testmanagement Enterprise Edition enables bulk actions on objects on list pages. Multiple objects may be deleted, duplicated, revisioned or displayed on a printer friendly page by selecting them and clicking a bulk action button.

1.3. Klaros-Testmanagement Mobile Edition Features

Additional features of Klaros-Testmanagement Mobile Edition:

Mobile Test Execution	With Klaros-Testmanagement Mobile Edition it is possible to execute manual tests on site, making use of the extra input facilities of a modern mobile device (smartphone or tablet).
Pausable Test Cases	Klaros-Testmanagement Mobile Edition allows pausing and resuming of tests.
Extended Recording of Test Results	Klaros-Testmanagement Mobile Edition can also use the sensors on a mobile device to record test results, i.e. the camera, microphone, GPS. The information captured with these sensors is linked to the test results and automatically saved on the server.
Network Independence	Klaros-Testmanagement Mobile Edition can also be used without an active network connection. Tests can be synchronized with the test management server at a later date.

More information can be found in the documentation of the Klaros-Testmanagement Mobile Edition Android Client.

Chapter 2. Introduction

2.1. What is Klaros-Testmanagement

Klaros-Testmanagement is a modern web application which helps you to organize and manage your test process. It documents all test relevant data by storing it in a database and gives an overview about the test progress of a project or piece of software at any time. A single item of test relevant data will be called *Artifact* throughout this document.

The main artifact is called *Project*. It contains all the other artifacts that are needed to define, execute and evaluate tests. For more details on defining projects see Section 6.1.

2.2. Test Cases - The Basis of Testing

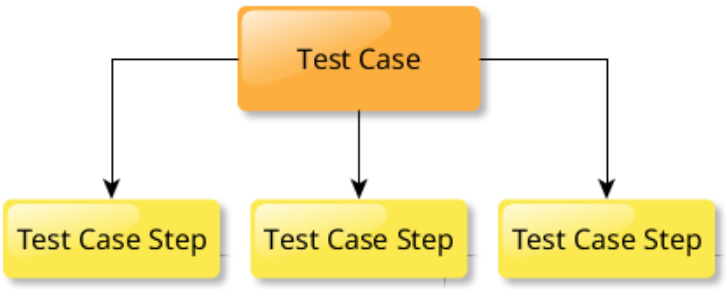


Figure 2.1. Test Case Structure

The central artifact of a test project is the *Test Case* which represents a single test. A test case can be executed in an automated or manual fashion. An automated test case refers to a piece of software code that gets executed and in return delivers a test result, usually in form of a test result file. A manual test case contains beside other entries one or more *Test Case Steps* which hold detailed written instructions and test conditions for the testers executing the test case.

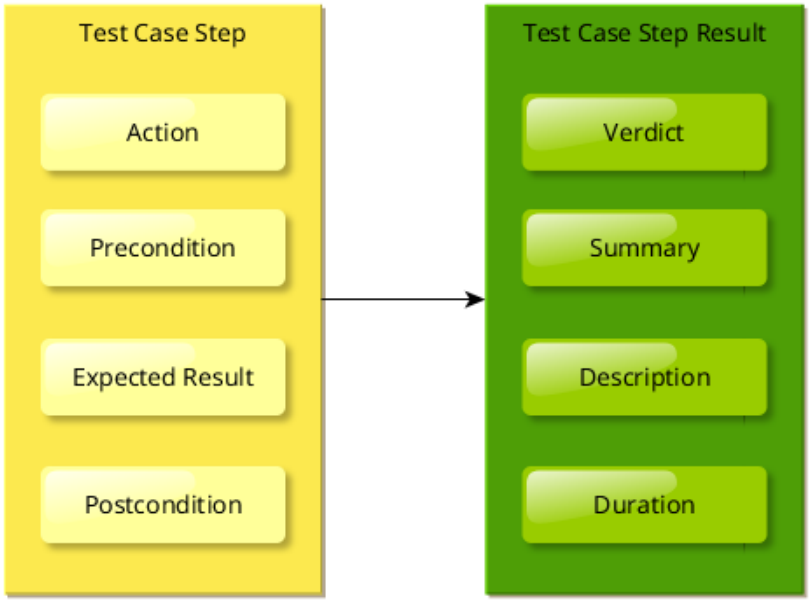


Figure 2.2. Test Case Step

A test case step consists of a **Precondition** (the state of the System under Test before the test starts), an **Action** (explains what the user has to do/test/try out). After the action the outcome should be the **Expected Result**, and a **Postcondition** (the state of the system under test after the action). Klaros-Testmanagement guides the user through the test case step by step. During the test execution every step will be assigned a **Test Case Step Result** which contains the verdict of the test case step. `Passed`, `Failed`, `Error` or `Skipped` are the possible verdicts which are available to the user.

If the precondition, postcondition or expected result does not match the description, the User should choose `Failure`. If an unexpected error in the system prevents the user from carrying out the step (for example, a button is missing) they should choose `Error`. If one of the test case step result verdicts is marked as `Error` or `Failure`, the whole test case result is also marked with that particular verdict. Only if every test case step is marked as passed, the encapsulating test case result is marked as passed.

Automated test cases usually have no test case steps because they are executed with an automated test program (e.g. JUnit). These test case results can be imported and stored with the other test related data into the database.

Requirements are another type of artifact in Klaros-Testmanagement which represent a condition or capability of the system under test. Each test case can relate to one or more requirements and vice-versa. More information regarding requirements is shown in (Section 2.7).

2.3. Test Suites - Organizing Test Cases

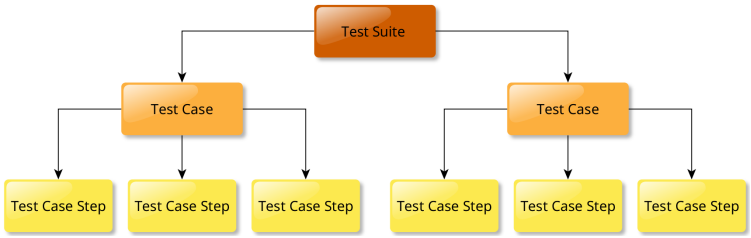


Figure 2.3. A Test Suite

Multiple test cases can be combined into another artifact that is called *Test Suite*. When executing a test suite, all test cases that are part of this test suite are executed in a row by the tester.

The same test cases can be part of multiple test suites and changes to a test case will automatically be reflected in all test suites referencing it.

2.4. Test Runs - Executed Tests with their Results

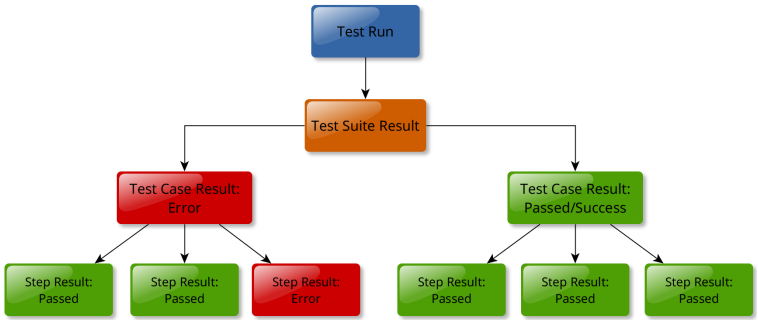


Figure 2.4. Test Run with Test Results

Every time a tester executes test cases or test suites, a *Test Run* containing the corresponding *Test Case Results* is created. The test run consists of all test case results created during the execution of a test case or test suite and stores additional information about the circumstances of the test run shown in the next section.

The simplest case is the execution of a single test case. This generates a test run with a single result (the test case result). If a test suite is executed, there will usually be more than one test result (one result for each test case). The test suite result lists all the test results of the executed test suite.

Test runs can be paused and continued at a later time. This means that it is possible that not all results that are stored in the database already have a known verdict, in this case the verdict is listed as `Unknown`.

2.5. Test Environment and System under Test - Influences of Test Results

Whenever a test case or test suite is executed, it needs to be executed in a defined *Test Environment*, and one executes it against a defined *System under Test*.

The result of a test is influenced by the version of the system being tested (system under test / SUT), since issues are detected in a particular software version, but may then be fixed in a later version.

Test environments represent external conditions which may have impact on the test result, like e.g. the operating system the system under test in running in. To document this properly, every test run is linked to a particular system under test and test environment.

2.6. Iterations - Subdividing your project into phases

An *Iteration* can group jobs, systems under test, test environments and requirements within a project for a selectable time frame. Every iteration represents a selected part of the project and its artifacts including all test results gathered during this phase. Grouping the artifacts like this makes it easier to identify different test cycles of a project. This allows for better integration of the test process especially with agile software development techniques.

2.7. Requirements – Meeting Quality Standards

To confirm if the system under test meets its quality standards it is needed to document which of the system requirements are covered by tests (*Coverage*) and how many of them have been executed with a passed result (*Compliance*).

To enable this traceability, it is needed to link requirements to one or more test cases and vice versa.

From this set of information Klaros-Testmanagement is able to automatically determine coverage and compliance information for the whole project or selected artifacts once the first test results are available.

2.8. Jobs - Planning the test process

Many users may execute tests in Klaros-Testmanagement simultaneously. To orchestrate this process it is recommended to plan the testing activities of the users. The planned execution of a test case or a test suite by a user is defined in the form of a *Job* (which is also an artifact). Jobs usually reference a test case or test suite. The intended test environment and system under test may be defined beforehand by the creator of a job. Jobs also contain a start date (when the testing should begin), a due date (a deadline for the execution of the test), and may be assigned a priority (how urgent the test execution is). Available priorities are: `Trivial`, `Minor`, `Major`, `Critical` and `Blocker`. Jobs may also contain status information like `Resolved`, `Reopened`, `Closed`, `In Progress` or `New` to document their progress.

After a job has been created it can be assigned to a user. This user then can execute the job and set its results. A job may be repeated and executed multiple times, so one job can refer to more than one test run.

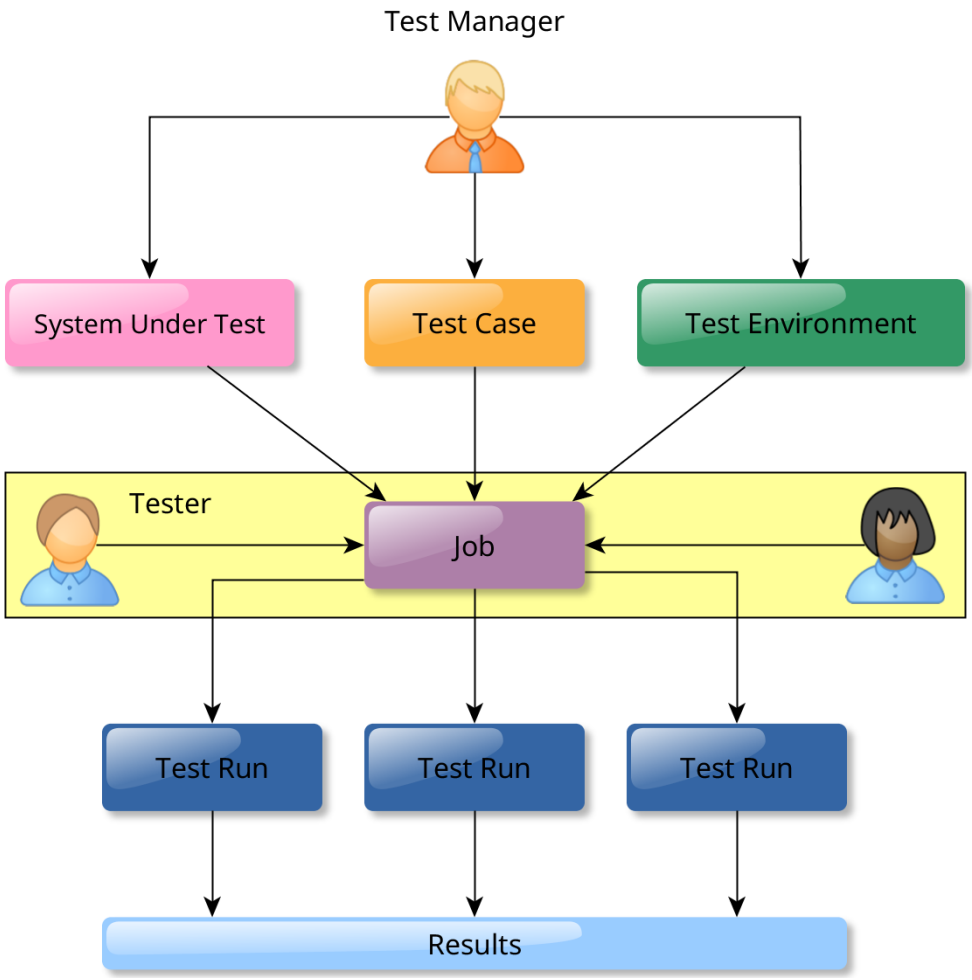


Figure 2.5. Jobs

In addition to test execution other job types exists allowing to schedule reviews of test cases or requirements and specify arbitrary tasks in a textual form.

Jobs can be nested and individually assigned to users. Therefore it is possible to track progress of larger, distributed test activities.

2.9. User Roles - Every User has his Rights

Klaros-Testmanagement defines four roles available to users: *Administrator*, *Manager*, *Tester* and *Guest*. Only a user with the `Administrator` role is able to create and assign other users.

`Guests` are only able to view artifacts and reports, they are not able to change anything in the system.

`Testers` have more permissions than those in the role of guest. They are not only able to view artifacts but in addition they are allowed to execute jobs, test cases and test suites and edit test results.

`Managers` inherit all rights of a tester and also possess additional rights to create and update artifacts like test cases, requirements and more.

`Administrators` have all available rights in the system. In addition to the rights a manager has they are able to create and delete projects and users as well as configuring system-wide parameters.

Each users has a defined default role which is assigned by an administrator when creating a user account. The `Manager`, `Tester` and `Guest` roles can also be assigned to users on a per-project basis providing a project based security scheme.

Detailed information about the role permissions can be found in Appendix A).

2.10. Overview of Artifacts

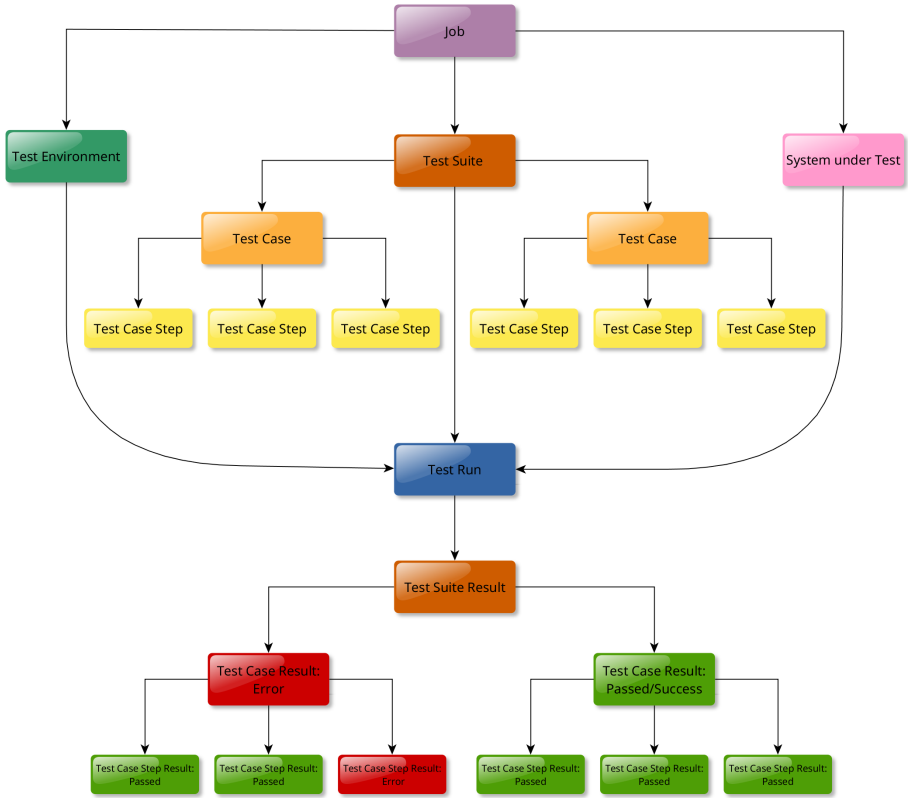


Figure 2.6. Artifacts Overview

2.11. Issue Management Systems

Klaros-Testmanagement can be integrated with issue tracking systems like Atlassian Jira, Bugzilla and many more (see Section 12.5). If an error of the system under test is detected in one of the test case steps, the user can easily create an entry in the issue tracking system from within Klaros-Testmanagement which is automatically linked and tracked inside both systems.

Chapter 3. Installation

Klaros-Testmanagement is equipped with an installer for Microsoft Windows- and Linux-based operating systems. Excluding a Java Runtime Environment, which is needed for the application and the installer itself, it will install all of the applications necessary to run Klaros-Testmanagement, including a file based database (Apache Derby) and an Apache Tomcat application server.

Warning

The installer configures Klaros-Testmanagement to use the Apache Derby database for persistence which is not intended for production use. For production systems, it is strongly recommended to use a database like MySQL, PostgreSQL, SQL-Server or Oracle Database which will provide significantly better performance. After running the initial installation process, the database can be changed (see Section 3.9, “Changing the Default Database”).

This chapter provides all information necessary to install Klaros-Testmanagement.

3.1. License Model

3.1.1. License Model (Users)

Depending on the version of Klaros-Testmanagement in use, a different user license model is applied. Since the Klaros-Testmanagement Community Edition has a smaller number of features in comparison to the other editions it supports an unlimited amount of users that can work in parallel. The Klaros-Testmanagement Enterprise Edition follows another license model. Each Klaros-Testmanagement Enterprise Edition license is bound to a set number of licensed users. This number dictates how many users can be active at any single point in time. If a user is to be created which would excess the number of licensed users, this user immediately gets deactivated thereafter. The user can later be re-activated if another active user gets deactivated first.

3.1.2. License Model (Mobile Devices)

The Klaros-Testmanagement Mobile Edition license is tied to a set number of mobile devices. If a mobile device connects to Klaros-Testmanagement for the first time and if there are no free mobile device slots available, then the mobile device gets deactivated. It can later be activated again if an administrator deactivates another mobile device first.

Important

If a user or mobile device gets deactivated, the only thing that happens is that subsequent logins are denied for that user. The complete history of that user as well as any (testing-)data related to that user is kept in the connected database.

3.2. Prerequisites

Since Klaros-Testmanagement is a web based application, the client and the server side have different installation prerequisites, which are explained in detail in the next sections.

3.2.1. Client Prerequisites

To access Klaros-Testmanagement, modern web browser supporting JavaScript is required.

Name	Version
Microsoft Internet Explorer	Version 7 and above
Mozilla Firefox	Version 3.5 and above
Google Chrome	Version 4 and above
Safari	Version 4 and above

Table 3.1. Supported Browsers

Please note that Konqueror is currently not a supported browser due to issues with the provided JavaScript engine.

Caution

Klaros-Testmanagement requires a minimum screen resolution of 1280x1024 pixels.

3.2.2. Server Prerequisites

To run the Klaros-Testmanagement server, a Microsoft Windows or Linux operating system running `Oracle`Java version 7 or 8 is required.

Warning

There are known issues when using alternative Java implementations like `GCJ` or `IcedTea/OpenJDK` which can per default be found in common Linux distributions. Using these implementations is strongly discouraged and not supported.

For optimal performance, using a separate database installation is highly recommended. This database is not required to run on the same physical machine as the Klaros-Testmanagement server.

Name	Description
Apache Derby	Version 10.10.1.1 and above (embedded and preconfigured)
Microsoft SQL-Server	Microsoft SQL Server 2012, SQL Server 2008 R2, SQL Server 2008

Name	Description
MySQL	Version 5.5 and above with InnoDB as the storage engine. MyISAM is not supported as the storage engine, since it doesn't provide referential integrity.
PostgreSQL	Version 8.4 and above
Oracle	Version 10g, 11g, 12c

Table 3.2. Supported Databases

3.2.2.1. Installing Java

If your system is not yet equipped with an Oracle Java 7 runtime environment, you need to download a JRE or JDK distribution from http://www.oracle.com/technetwork/java/javase/downloads/index.html. Both JRE and JDK installations are supported. For maximum performance choose a distribution matching your system architecture (x86/x64).

Alternative Java-Runtime Environments

Out of the box, most Linux distributions do not come with an Oracle Java installation but some alternative Java implementation like GCJ or OpenJDK. Klaros-Testmanagement is not supported in such an environment as there are **known issues** with some of these implementations.

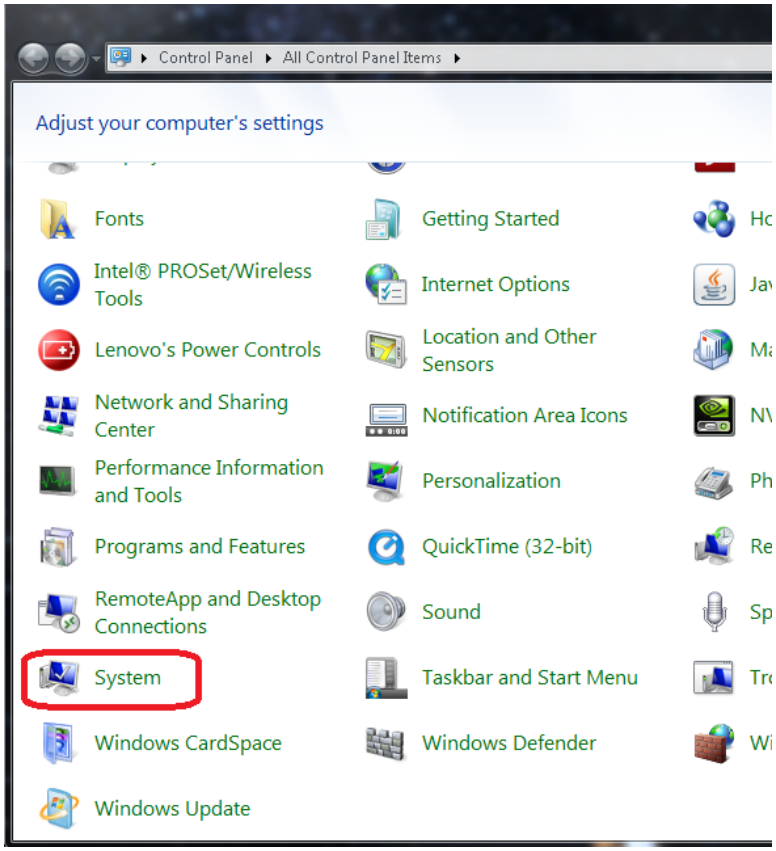
Make sure that the JAVA_HOME/JRE_HOME environment variable is set!

A common caveat is that although Java is installed on your machine, the installer or launch script will complain that neither the JAVA_HOME nor the JRE_HOME environment variable is defined. Windows users will find an option to set this in their computer control panel. The instructions to reach this vary with different versions of windows, an example is shown below. Linux users should set these variables in their startup scripts.

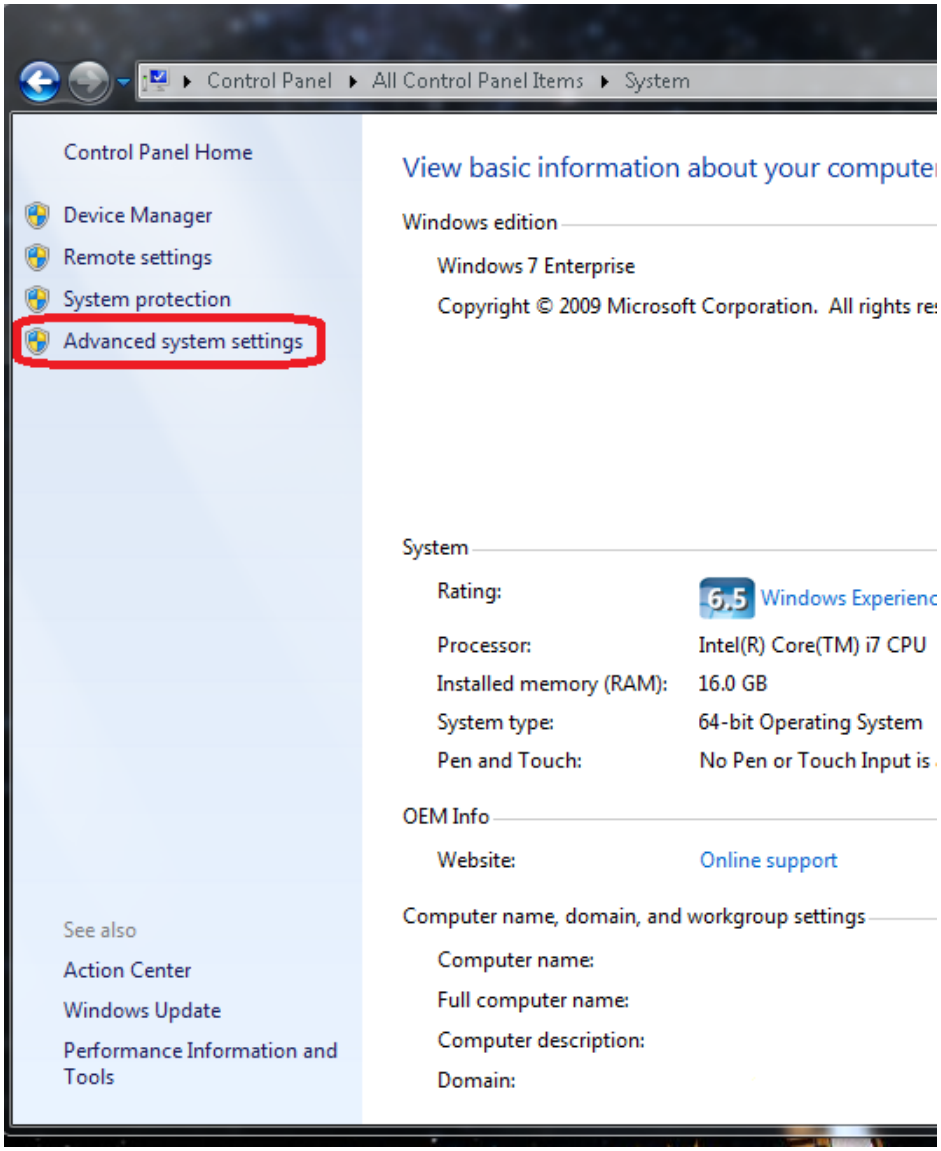
If you have installed a Java SDK you would set the JAVA_HOME variable. If your system only contains a Java Runtime (JRE) please use JRE_HOME accordingly.

3.2.2.2. Setting the JAVA_HOME/JRE_HOME environment variable under Windows

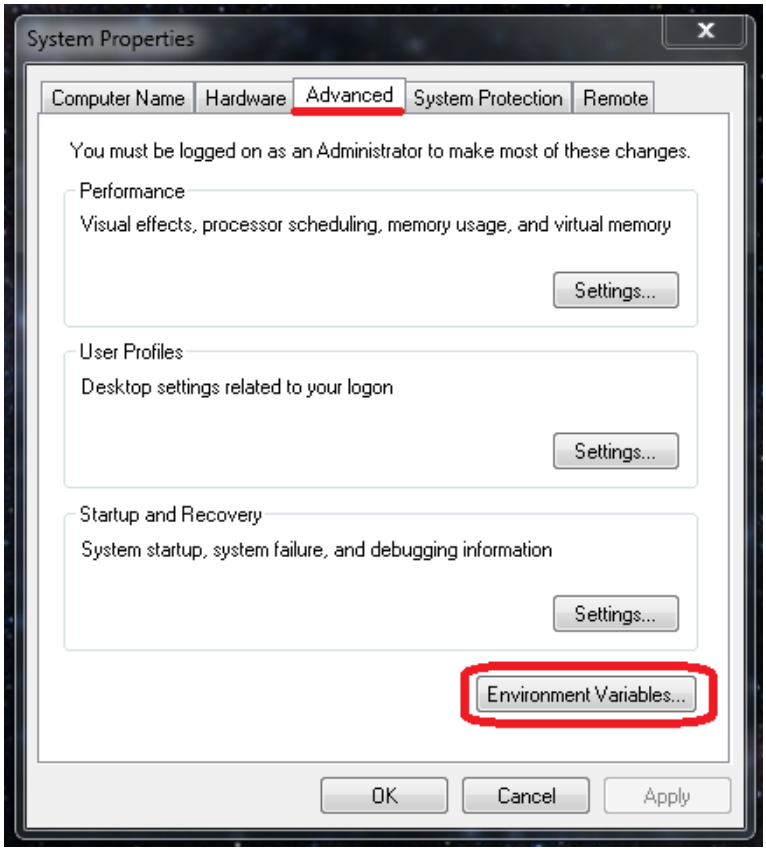
- Go to *Control Panel*.
- Find System.



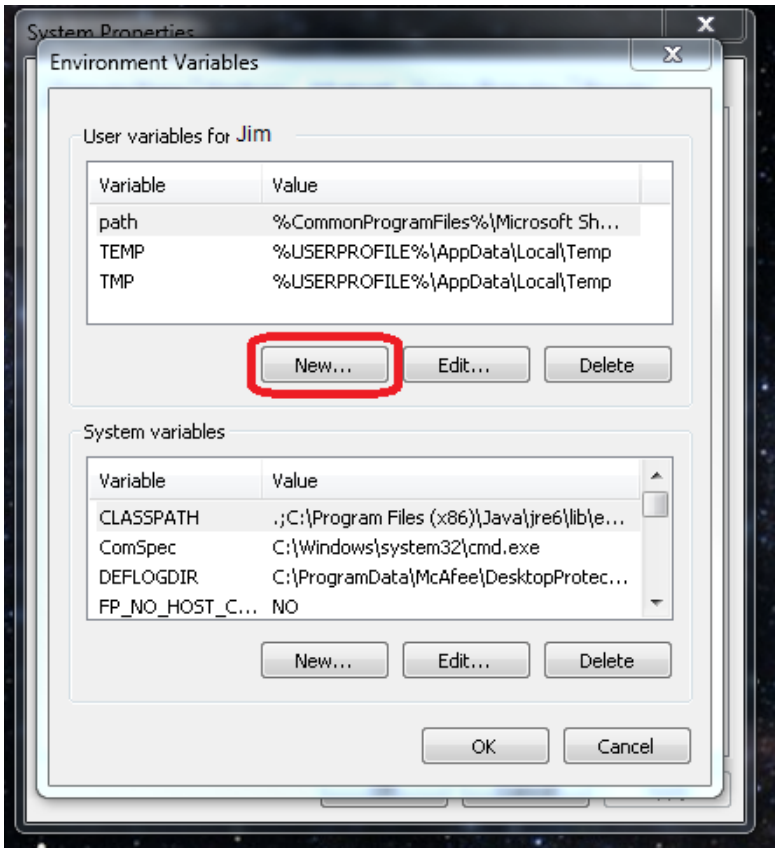
- Then go to *Advanced system settings*



- Click on the panel *Advanced* then the button *Environment Variables*

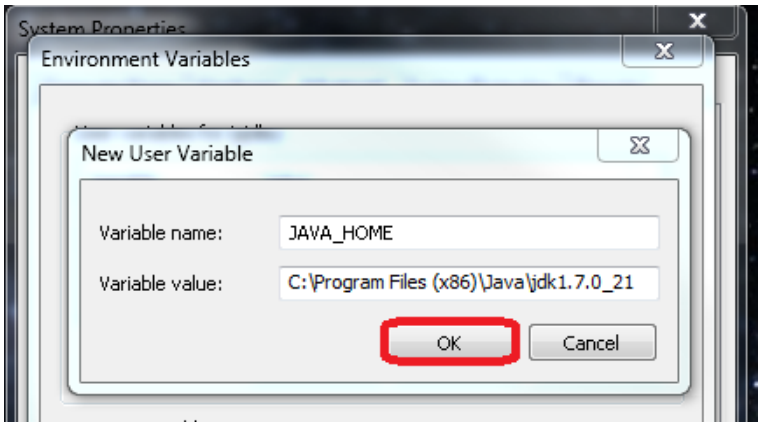


- Then this Window should pop up.



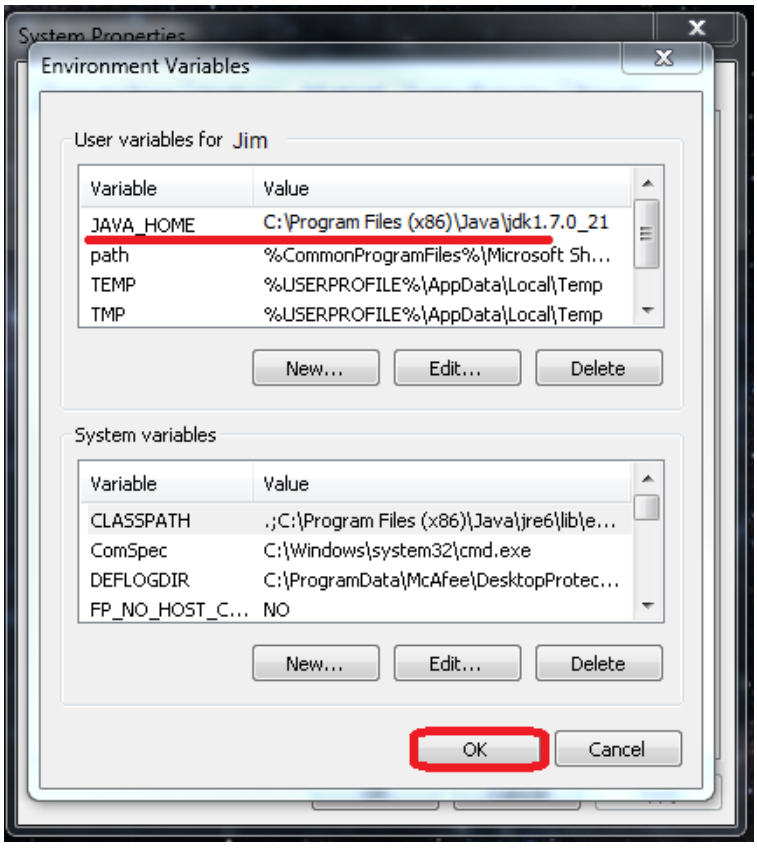
If the variable `JAVA_HOME` is not listed, press **New**. Otherwise make sure the variable value of `JAVA_HOME` is still correct.

If the name of the variable is `JAVA_HOME`, the value must be the absolute path of the Java JDK. You can usually find the JDK in `C:\Program Files\Java\jdk1.7.0_nn`, where *nn* is the patch level of the JDK. Java JREs can be found in `C:\Program Files\Java\jre7`. Add `(x86)` to `C:\Program Files` if you use a 32bit Java on a 64bit operating system, which is not recommended for production use).



After pressing *OK* the Java variable is set.

In the end, it should look something like this:



3.2.2.3. Microsoft Windows

To run Klaros-Testmanagement on Microsoft Windows operating systems the following requirements should be met:

- **Minimum requirements**

  2GB RAM, 2GHz Single Core CPU

- **Recommended requirements (Remote Database)**

  4GB RAM, 3GHz Dual-Core CPU

- **Recommended requirements (Local Database)**

  8GB RAM, 3GHz Quad-Core CPU

- **Operating system**

  Microsoft Windows XP, Microsoft Windows Server 2003, Microsoft Windows Vista, Microsoft Windows Server 2008, Microsoft Windows 7, Microsoft Windows Server 2012, Microsoft Windows 8 or Microsoft Windows 8.1.

- **Java Runtime Environment**

  Oracle Java Runtime Platform, Standard Edition 7 or 8

3.2.2.4. Linux

To run Klaros-Testmanagement on Linux operating systems the following requirements should be met:

- **Minimum requirements**

  2GB RAM, 2GHz Single Core CPU

- **Recommended requirements (Remote Database)**

  4GB RAM, 3GHz Dual-Core CPU

- **Recommended requirements (Local Database)**

  8GB RAM, 3GHz Quad-Core CPU

- **Operating system**

  Linux IA32 (Intel 32 bit architecture) or Intel64/x86-64 (Intel 64 bit architecture) distribution containing Version 2.2.1 or better of the GTK+ widget toolkit and associated libraries.

Note

The GTK+ libraries are only needed to run the Klaros-Testmanagement installer in windowed mode. Running the installer in console mode will work without this requirement.

- **Java Runtime Environment**

  Oracle Java Platform, Standard Edition 7 or 8

- **Open file limitation**

Caution

Klaros-Testmanagement Enterprise Edition stores database search indices in the file system in the `.klaros/indexes` folder in home directory of the user account that is running the Klaros-Testmanagement application server. During the indexing process the amount of open files may exceed the predefined limit in the Linux operating system. The command **ulimit -n** shows the configured number of open files allowed, which should be set to a value of at least 4096. Please consult your system documentation on how to set this value (Usually by editing `/etc/security/limits.conf`).

3.2.2.5. Disc Space

Klaros-Testmanagement stores binary attachments in a file based repository inside the `.klaros` folder in home directory of the user account that is running the Klaros-Testmanagement application server. Depending on the usage pattern a sufficient amount of disk space should be reserved for this account. If the file based Derby database is active, the database files will per default also be located in the user home directory.

Note

Reserving 10GB of disc space should be enough for typical usage patterns.

3.3. Installation Process

Under Windows the installer is invoked by running **Klaros-Setup.exe** .

Under Linux the installer can be started by entering **java -jar Klaros-Setup.jar** in a user shell.

The following screens show each step involved in installing Klaros-Testmanagement.

3.3.1. Step 1: Welcome

The initial step shows a welcome screen to the user (Figure 3.1).

Figure 3.1. Welcome Screen

The installation may be aborted by clicking the **Quit** button. Clicking **Next** proceeds with the installation.

3.3.2. Step 2: Information

The second step shows information about the product and the revision history, listing the fixed issues and the newly added features (Figure 3.2).

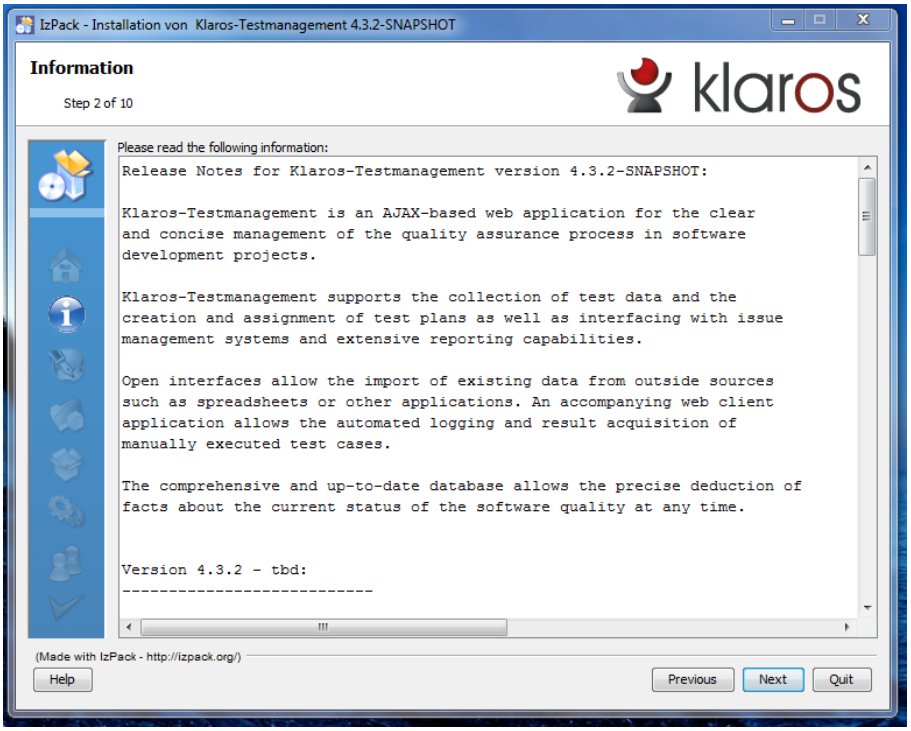

IzPack - Installation von Klaros-Testmanagement 4.3.2-SNAPSHOT
Information
Step 2 of 10
klaros
Please read the following information:
Release Notes for Klaros-Testmanagement version 4.3.2-SNAPSHOT:
Klaros-Testmanagement is an AJAX-based web application for the clear
and concise management of the quality assurance process in software
development projects.
Klaros-Testmanagement supports the collection of test data and the
creation and assignment of test plans as well as interfacing with issue
management systems and extensive reporting capabilities.
Open interfaces allow the import of existing data from outside sources
such as spreadsheets or other applications. An accompanying web client
application allows the automated logging and result acquisition of
manually executed test cases.
The comprehensive and up-to-date database allows the precise deduction of
facts about the current status of the software quality at any time.
Version 4.3.2 - tbd:
(Made with IzPack - http://izpack.org/)
Help
Previous
Next
Quit

Figure 3.2. Information Screen

The installation may be aborted by clicking the **Quit** button. Clicking **Previous** goes back to the *Welcome* step and clicking **Next** proceeds with the installation.

3.3.3. Step 3: Licensing Agreements

The third step shows information about the license agreement for Klaros-Testmanagement. The license must be accepted to continue the installation. (Figure 3.3).

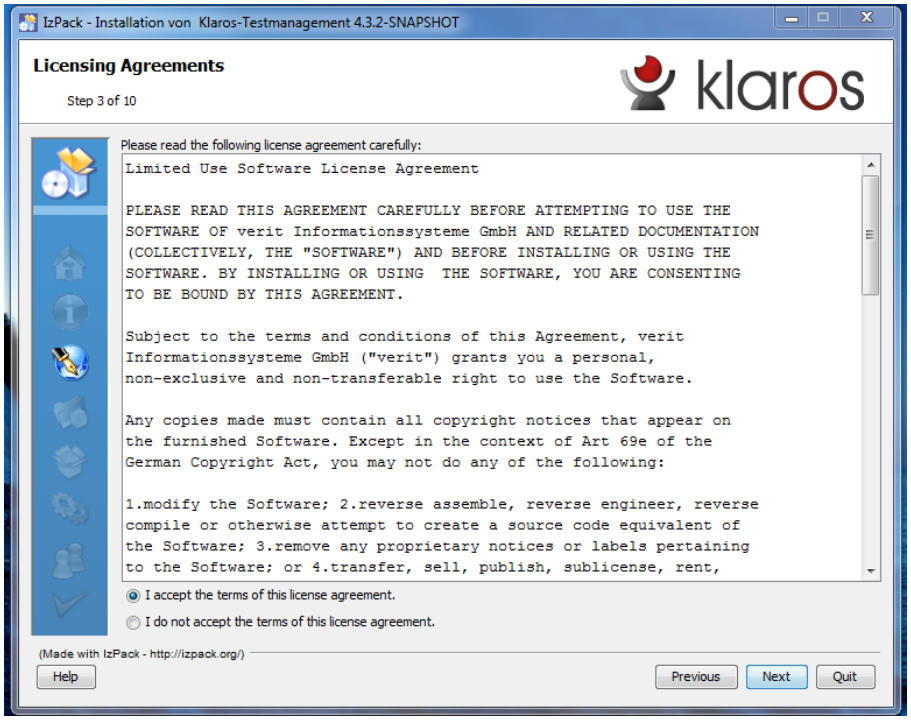

IzPack - Installation von Klaros-Testmanagement 4.3.2-SNAPSHOT
Licensing Agreements
Step 3 of 10
klaros
Please read the following license agreement carefully:
Limited Use Software License Agreement
PLEASE READ THIS AGREEMENT CAREFULLY BEFORE ATTEMPTING TO USE THE SOFTWARE OF verit Informationssysteme GmbH AND RELATED DOCUMENTATION (COLLECTIVELY, THE "SOFTWARE") AND BEFORE INSTALLING OR USING THE SOFTWARE. BY INSTALLING OR USING THE SOFTWARE, YOU ARE CONSENTING TO BE BOUND BY THIS AGREEMENT.
Subject to the terms and conditions of this Agreement, verit Informationssysteme GmbH ("verit") grants you a personal, non-exclusive and non-transferable right to use the Software.
Any copies made must contain all copyright notices that appear on the furnished Software. Except in the context of Art 69e of the German Copyright Act, you may not do any of the following:
1.modify the Software; 2.reverse assemble, reverse engineer, reverse compile or otherwise attempt to create a source code equivalent of the Software; 3.remove any proprietary notices or labels pertaining to the Software; or 4.transfer, sell, publish, sublicense, rent,
I accept the terms of this license agreement.
I do not accept the terms of this license agreement.
(Made with IzPack - http://izpack.org/)
Help
Previous
Next
Quit

Figure 3.3. Licensing Agreement Screen

The installation may be aborted by clicking the **Quit** button. Clicking **Previous** goes back to the *Information* step and clicking **Next** proceeds with the installation.

3.3.4. Step 4: Target Path

The fourth step requests the target path where Klaros-Testmanagement will be installed. The user can use the **Browse** button to search for the specific path in the local file system (Figure 3.4).

Important

It is not recommended to install Klaros-Testmanagement in the Programs folder on systems running Microsoft Windows Vista or later. By default, Klaros-Testmanagement will not be able to start if installed here, as tomcat requires write access to the install folder in order to deploy the web application. This is blocked by Microsoft Windows User Account Control (UAC).

If you wish to install Klaros-Testmanagement in this location, UAC must be disabled.

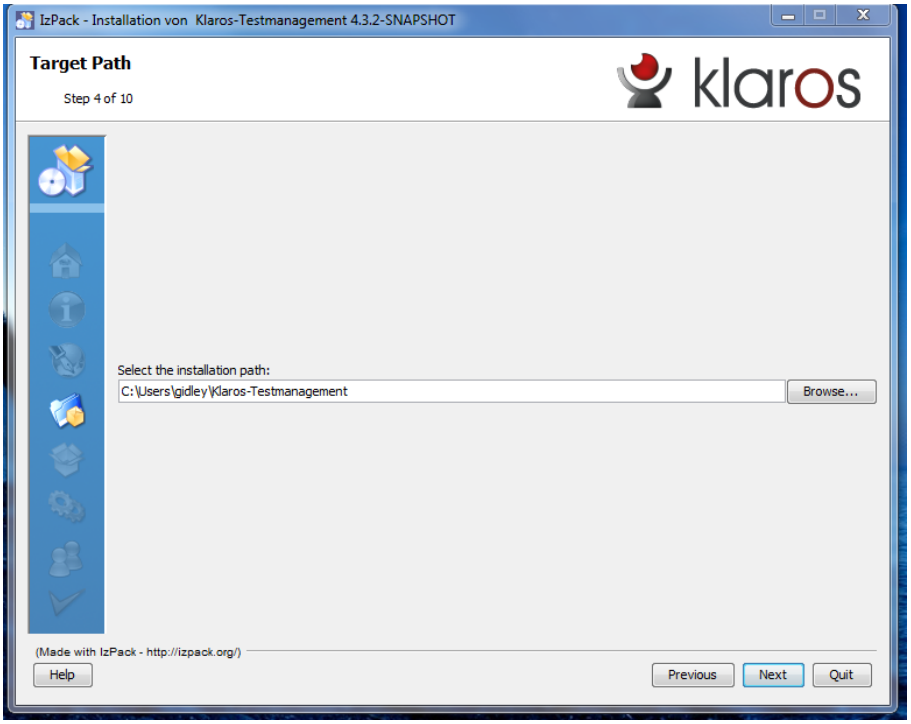


Figure 3.4. Target Path Screen

The installation may be aborted by clicking the **Quit** button. Clicking **Previous** goes back to the *Licensing Agreement* dialog and clicking **Next** proceeds with the installation.

3.3.5. Step 5: Select Installation Packages

The fifth step allows to select the packages that are installed with Klaros-Testmanagement.

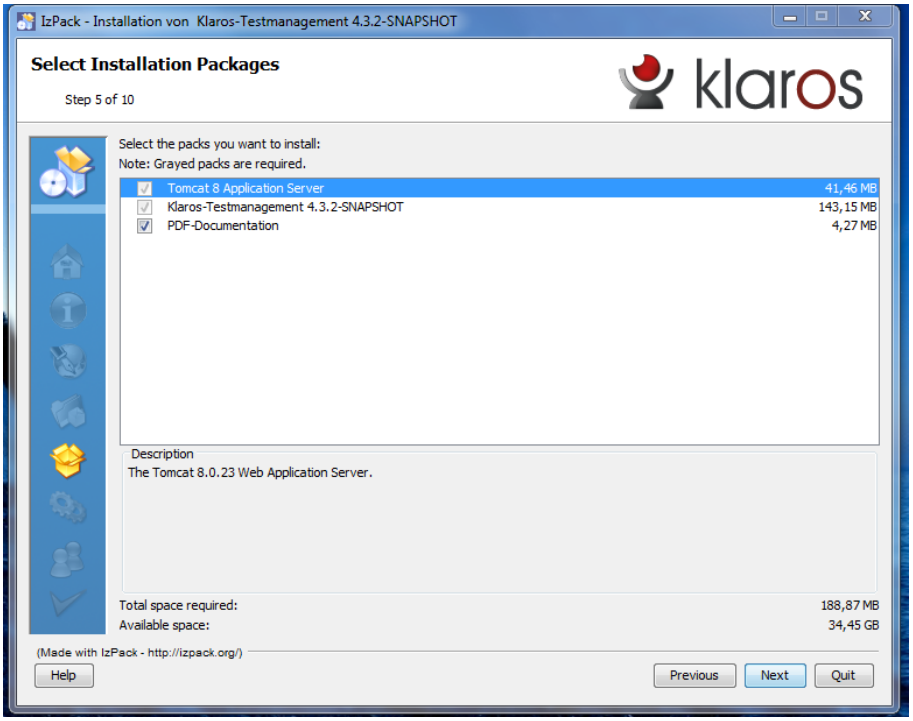


Figure 3.5. Select Installation Packages Screen

The installation may be aborted by clicking the **Quit** button. Clicking **Previous** goes back to the *Target Path* dialog, clicking **Next** proceeds with the installation.

3.3.6. Step 6: User Data

In the sixth step the Tomcat server settings can be changed.

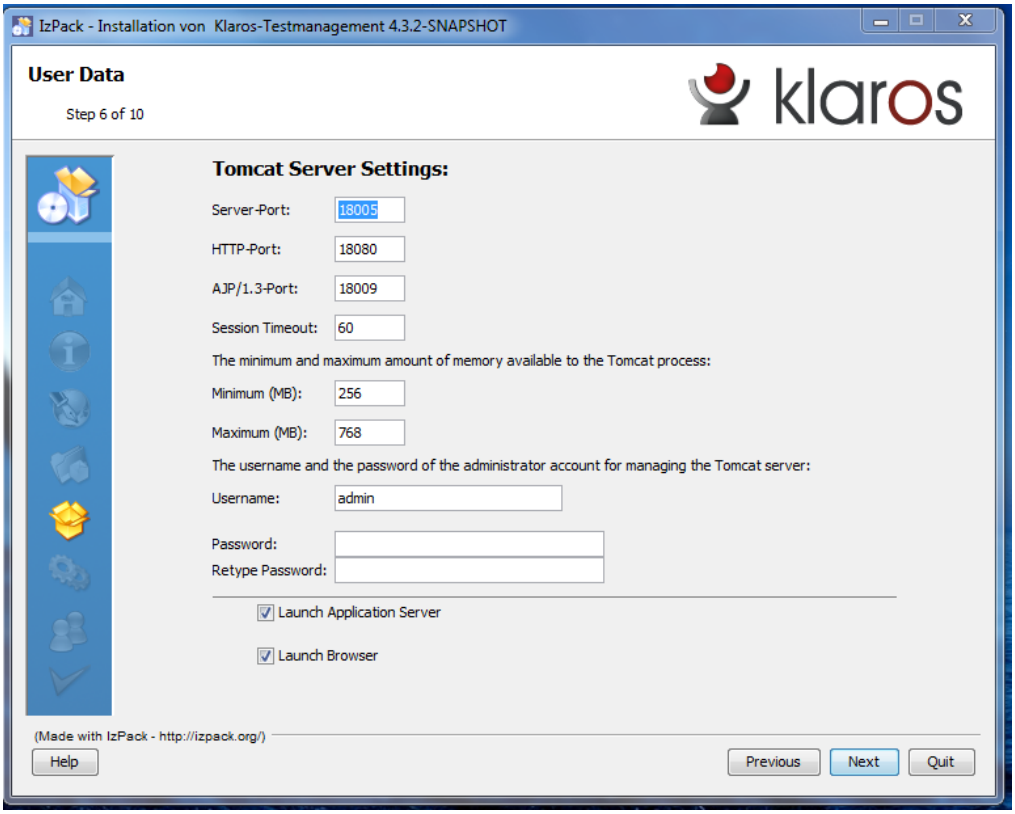


Figure 3.6. User Data Screen

It is possible to set the ports used by Tomcat(Server-Port, HTTP-Port and AJP/1.3-Port), the minimum and maximum amount of memory available to the Tomcat process as well as the user name and password of the Tomcat application server administrator.

In addition it is possible to control the startup of the application during the installation. If the *Launch Application Server* checkmark is set, the application server will be launched automatically during the installation process. If the *Launch Browser* checkmark is set, a native web browser instance will be redirected to the application login page. Both of these actions happen during the Section 3.3.8, “Step 8: Perform External Processes” phase.

The installation may be aborted by clicking the **Quit** button. Clicking **Previous** button goes back to the *Select Installation Packages* step and clicking **Next** button proceeds with the installation.

3.3.7. Step 7: Installation

The seventh step starts the installation of Klaros-Testmanagement and shows its progress. The following screenshots show the *Installation in Progress* Screen and the *Installation finished* screen (Figure 3.7).

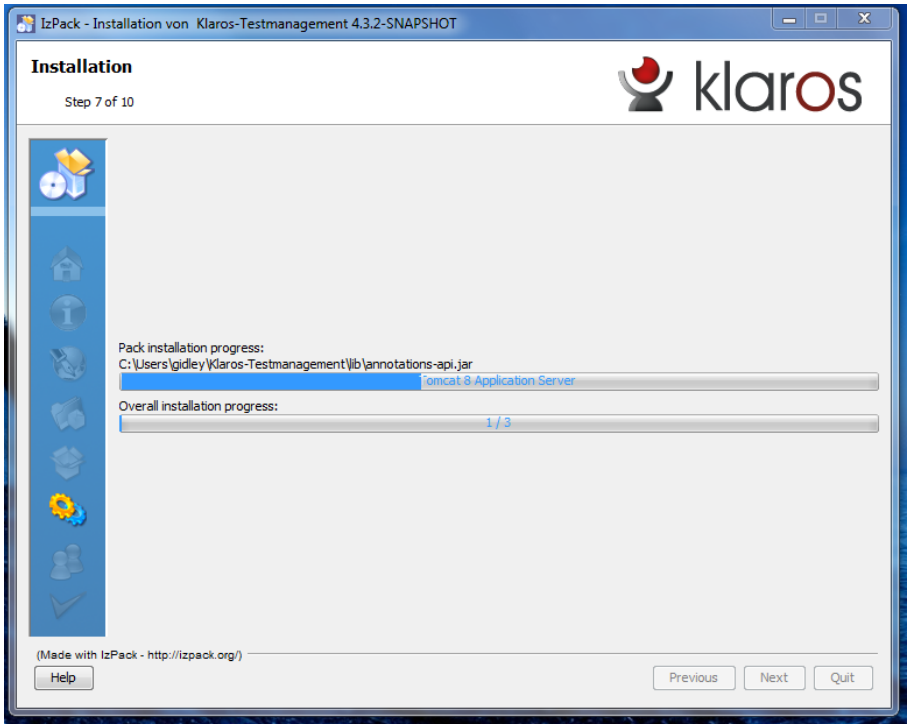


Figure 3.7. Installation in Progress Screen

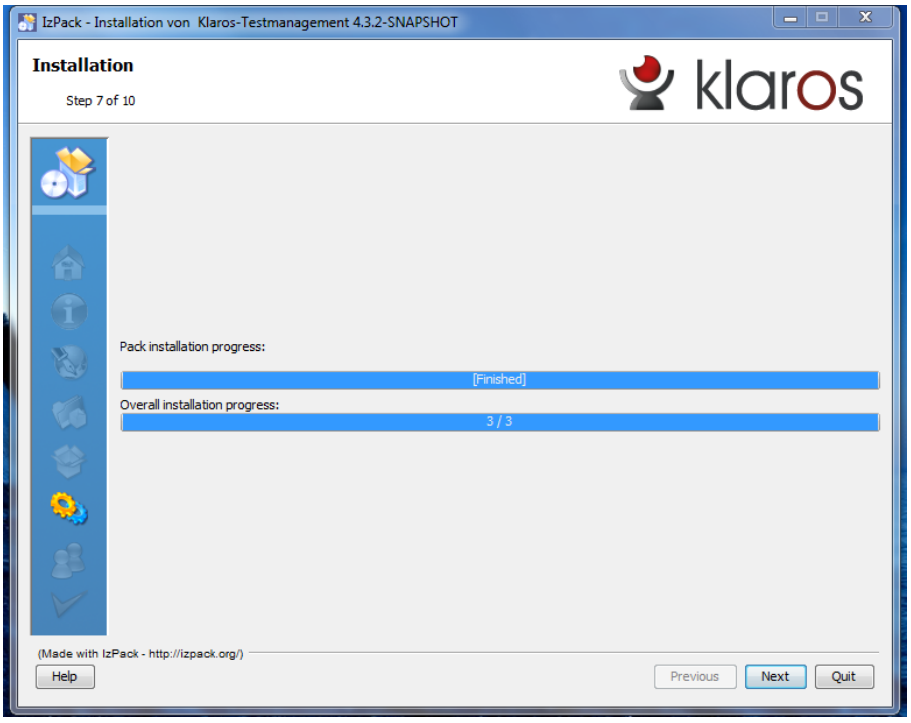


Figure 3.8. Installation Finished Screen

The installation may be aborted by clicking the **Quit** button. Clicking **Next** proceeds with the installation.

3.3.8. Step 8: Perform External Processes

The optional eighth step performs external processes such as setting the environment variables and starting the Tomcat service. In case none of the flags *Launch Application Server* and *Launch Browser* are checked, this panel will be skipped.

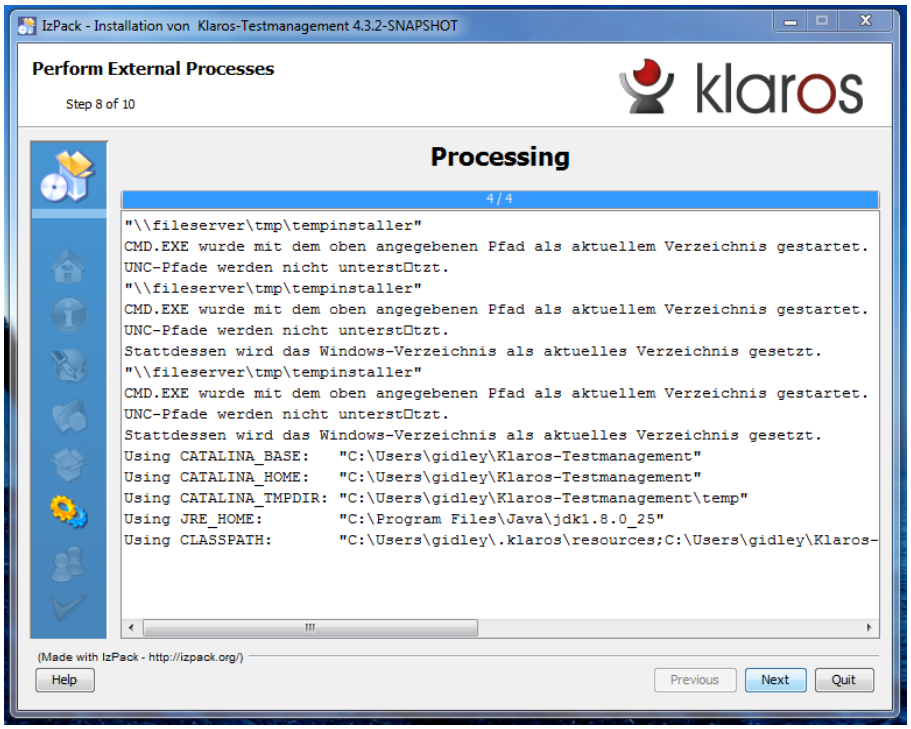

IzPack - Installation von Klaros-Testmanagement 4.3.2-SNAPSHOT
Perform External Processes
Step 8 of 10
klaros
Processing
4 / 4
"\\fileserver\tmp\tempinstaller"
CMD.EXE wurde mit dem oben angegebenen Pfad als aktuellem Verzeichnis gestartet.
UNC-Pfade werden nicht unterstützt.
"\\fileserver\tmp\tempinstaller"
CMD.EXE wurde mit dem oben angegebenen Pfad als aktuellem Verzeichnis gestartet.
UNC-Pfade werden nicht unterstützt.
Stattdessen wird das Windows-Verzeichnis als aktuelles Verzeichnis gesetzt.
"\\fileserver\tmp\tempinstaller"
CMD.EXE wurde mit dem oben angegebenen Pfad als aktuellem Verzeichnis gestartet.
UNC-Pfade werden nicht unterstützt.
Stattdessen wird das Windows-Verzeichnis als aktuelles Verzeichnis gesetzt.
Using CATALINA_BASE: "C:\Users\gidley\Klaros-Testmanagement"
Using CATALINA_HOME: "C:\Users\gidley\Klaros-Testmanagement"
Using CATALINA_TMPDIR: "C:\Users\gidley\Klaros-Testmanagement\temp"
Using JRE_HOME: "C:\Program Files\Java\jdk1.8.0_25"
Using CLASSPATH: "C:\Users\gidley\.klaros\resources;C:\Users\gidley\Klaros-
(Made with IzPack - http://izpack.org/)
Help
Previous
Next
Quit

Figure 3.9. Perform External Processes Screen

The installation may be aborted by clicking the **Quit** button. Clicking **Next** proceeds with the installation.

3.3.9. Step 9: Setup Shortcuts

The ninth step requests the options for shortcuts. The installer can create shortcuts in the Start-Menu and on the desktop (Figure 3.10).

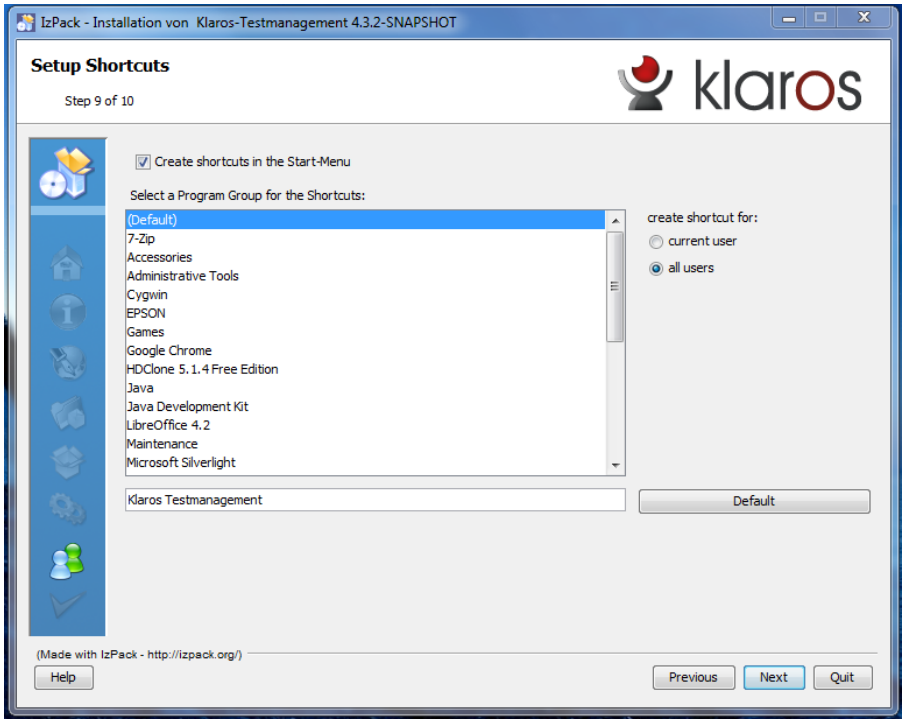


Figure 3.10. Setup Shortcuts Screen

This step looks different on Linux installations but has the same functionality.

The installation may be aborted by clicking the **Quit** button. Clicking **Previous** goes back to the *Perform External Processes* step and clicking **Next** proceeds with the installation.

3.3.10. Step 10: Installation Finished

The tenth step notifies the user that Klaros-Testmanagement was installed successfully and shows the path to the uninstaller program (Figure 3.11).

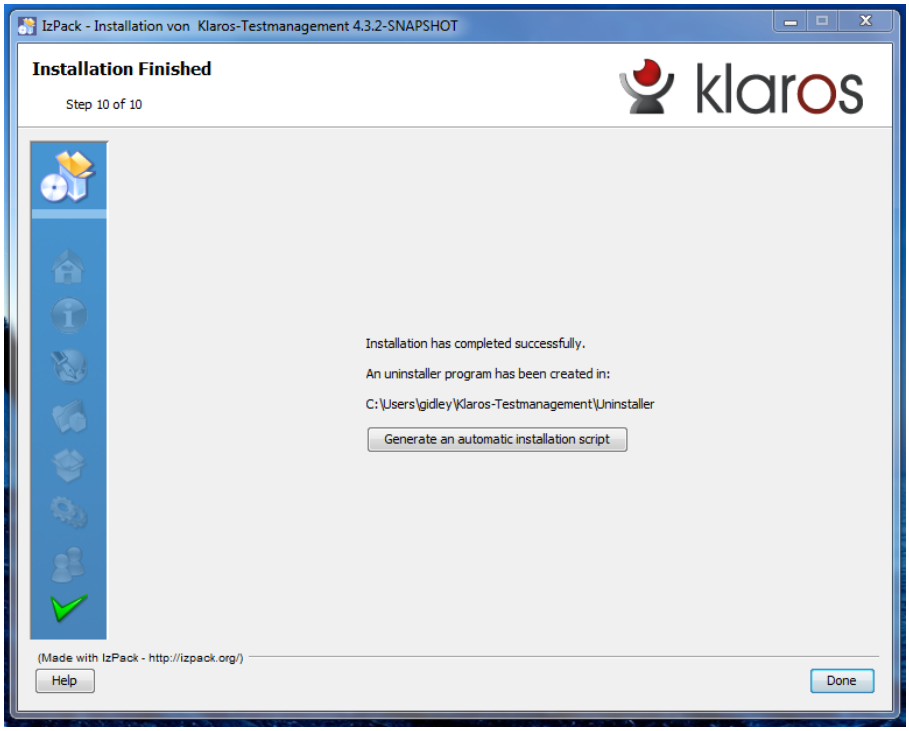


Figure 3.11. Installation Finished Screen

The **Generate an automatic installation script** button saves a complete script of the installation with the selected user choices, so that the same installation could be repeated unattended or on other machines.

The installation can be completed by clicking the **Done** button.

3.4. Console based installation

The Klaros-Testmanagement installer also supports a GUI-less variant which only needs a shell/command window and can be invoked as follows:

java -jar Klaros-Setup.jar -console

This displays the content of the installer panels in the shell window and lets the user input her choices via keyboard.

The UI controls are replaced by short key sequences:

Enter	shows the next page of a text panel or accepts a default value
1+**Enter**	selects a checkbox
0+**Enter**	deselects a checkbox
n+**Enter**	chooses the nth item from a list

3.4.1. Step 1: Welcome

The Welcome Panel (Section 3.3.1, “Step 1: Welcome”) is displayed and wants to be committed by **1**+**Enter**.

```
Please read the following information:


Version 4.3.0
(c)
Copyright 2009-2015 verit Informationssysteme GmbH. All Rights Reserved.


Press 1 to continue, 2 to quit, 3 to redisplay
1
```

3.4.2. Step 2: Information

The Information Panel (Section 3.3.2, “Step 2: Information”) needs some paging. Just press **Enter** to see the next page, until a choice is requested.

```
Please read the following information:
Release Notes for Klaros-Testmanagement version 4.3.0:
Klaros-Testmanagement is an AJAX-based web application for the clear
and concise management of the quality assurance process in software
development projects.

...

Press Enter to continue, X to exit

...

Additional information can be found at: http://www.klaros-testmanagement.com/
verit Informationssysteme GmbH
67657 Kaiserslautern
Europaallee 10
Germany

Web: http://www.verit.de/
E-Mail: support@verit.de
Phone: +49 631 520 840 00
Fax: +49 631 520 840 01

Press 1 to continue, 2 to quit, 3 to redisplay
1
```

3.4.3. Step 3: Licensing Agreements

The License Panel (Section 3.3.3, “Step 3: Licensing Agreements”) can be paged with a repetition of **Enter**. Commit with **1**+**Enter**.

```
Limited Use Software License Agreement
PLEASE READ THIS AGREEMENT CAREFULLY BEFORE ATTEMPTING TO USE THE
SOFTWARE OF verit Informationssysteme GmbH AND RELATED DOCUMENTATION
(COLLECTIVELY, THE "SOFTWARE") AND BEFORE INSTALLING OR USING THE
SOFTWARE. BY INSTALLING OR USING THE SOFTWARE, YOU ARE CONSENTING
TO BE BOUND BY THIS AGREEMENT.

...

limitations may not apply to you. You may also have other rights
```

```
that vary from state to state.

Press 1 to accept, 2 to reject, 3 to redisplay
1
```

3.4.4. Step 4: Target Path

The Target Path Panel (Section 3.3.4, “Step 4: Target Path”) looks like this with the console installer:

```
Select target path [/opt/klaros-testmanagement]
```

You can press **Enter** to accept the proposed path or type another. The path will be created, if it did not already exist. If the directory already exists, a warning is displayed and you can cancel the installation or continue.

3.4.5. Step 5: Select Installation Packages

Select Installation Packages (Section 3.3.5, “Step 5: Select Installation Packages”) is more complex in console mode. The required packages (Tomcat 8 and Klaros-Testmanagement) are displayed for information only and the request targets the optional PDF-Documentation. Typing **Y** chooses to install the PDF-Documentation, where **N** omits it. After this choice you are requested to commit the selection by typing **1** and then **Enter**.

```
Tomcat 7 Application Server required
Klaros-Testmanagement required
PDF-Documentation
Enter Y for Yes, N for No: N
Done!

Press 1 to continue, 2 to quit, 3 to redisplay
1
```

3.4.6. Step 6: User Data

After that, the installer requests user data (Section 3.3.6, “Step 6: User Data”). The installer presents default values which can be comitted by pressing **Enter**. You can enter a different value in case the default is not suitable.Checkboxes show [x] if the checkbox is set, [ ] otherwise. Pressing **Enter**accepts the current value, which can be set and unset explicitly with **1**and **0**respectively.

In this example we adopt most of the default values. Only the memory settings and the tomcat admin password (withno default) are entered and the two flags are unset:

```
Tomcat Server Settings:

Server-Port: [18005]
HTTP-Port: [18080]
AJP/1.3-Port: [18009]
Session Timeout: [60]

The minimum and maximum amount of memory available to the Tomcat process:
Minimum (MB): [256] 400
Maximum (MB): [512] 1024
```

```
The username and the password of the administrator account for managing the Tomcat server:
Username: [admin]

Password:beSureNotToUseThisPassword
Retype Password:beSureNotToUseThisPassword

-----------------------------------------
  [x] Launch Application Server
Enter 1 to select, 0 to deselect:
0

  [x] Launch Browser
Enter 1 to select, 0 to deselect:
0


Press 1 to continue, 2 to quit, 3 to redisplay
1
```

3.4.7. Step 7: Installation

Packages are displayed as soon as they are installed (Section 3.3.7, “Step 7: Installation”).

```
[ Starting to unpack ]
[ Processing package: Tomcat 7 Application Server (1/2) ]
[ Processing package: Klaros-Testmanagement (2/2) ]
[ Unpacking finished ]
```

3.4.8. Step 8: Perform External Processes

This panel is skipped since we chose not to launch the server or a browser, otherwise startup messages of the server could be seen here. See Section 3.3.8, “Step 8: Perform External Processes” on how this is handled in GUI mode.

3.4.9. Step 9: Setup Shortcuts

In a GUI-less environment we do not want any menu entries or icons on desktops, thus we choose **N**. See Section 3.3.9, “Step 9: Setup Shortcuts” for details on shortcut setup in windowed environments.

```
Create shortcuts in the Start-Menu
Enter Y for Yes, N for No: N
```

3.4.10. Step 10: Installation Finished

The generation of an automatic installation scripts does not work in console mode, so please choose **N** here. Section 3.3.10, “Step 10: Installation Finished” shows how the installation finishes in GUI mode, including the optional creation of automatic installation scripts. You can create such a script by installing Klaros-Testmanagement on a server with a GUI and modifying the script to fit your needs. Automatic installation scripts will work for unattended installations in console mode as well.

```
Generate an automatic installation script
```

```
   Enter Y for Yes, N for No: N
   Installation was successful
   application installed on /opt/klaros-testmanagement
   [ Writing the uninstaller data ... ]
   [ Console installation done ]
```

After the installation has finished, you can find the installation of Klaros-Testmanagement in the folder `/opt/klaros-testmanagement`. See Section 3.10, "Installing Klaros-Testmanagement as a Service" for details on how to configure Klaros-Testmanagement to run as a service.

3.5. Automated Installation Script

If you've previously installed Klaros-Testmanagement and chose to save an automated installation script, you can install Klaros-Testmanagement again with the same parameters using one of the following commands. This example assumes the automated installation script was saved as `auto-install.xml`.

Linux: **java -jar Klaros-Setup.jar -console auto-install.xml**

Windows: **Klaros-Setup.exe auto-install.xml**

In case you cannot run the Klaros-Testmanagement installer on a system with graphical user interface, you can build an automated installation script and run an unattended installation. Use the following sample script and adapt installpath and the userInputs. The installation may issue some warnings of the form `AutomationHelper class not found for panel ...`, which can be safely ignored.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AutomatedInstallation langpack="eng">
  <com.izforge.izpack.panels.htmlhello.HTMLHelloPanel
    id="hello" />
  <com.izforge.izpack.panels.htmlinfo.HTMLInfoPanel
    id="info" />
  <com.izforge.izpack.panels.licence.LicencePanel
    id="license" />
  <com.izforge.izpack.panels.target.TargetPanel
    id="target">
    <installpath>/opt/Klaros-Testmanagement</installpath>
  </com.izforge.izpack.panels.target.TargetPanel>
  <com.izforge.izpack.panels.packs.PacksPanel id="packs">
    <pack index="0" name="Tomcat 8 Application Server" selected="true" />
    <pack index="1" name="Klaros-Testmanagement" selected="true" />
    <pack index="2" name="PDF-Documentation" selected="false" />
  </com.izforge.izpack.panels.packs.PacksPanel>
  <com.izforge.izpack.panels.userinput.UserInputPanel
    id="portselection">
    <userInput>
      <entry key="tomcat_server_port" value="18005" />
      <entry key="tomcat_http_port" value="18080" />
      <entry key="tomcat_https_port" value="18443" />
      <entry key="tomcat_ajp_port" value="18009" />
      <entry key="tomcat_session_timeout" value="60" />
      <entry key="tomcat_memory_min" value="256" />
      <entry key="tomcat_memory_max" value="512" />
      <entry key="tomcat_admin_name" value="tcadmin" />
      <entry key="tomcat_admin_password" value="IfYouDontChangeThisDontBlameMe" />
      <entry key="start_server_flag" value="false" />
```

```
      <entry key="launch_browser_flag" value="false" />
    </userInput>
  </com.izforge.izpack.panels.userinput.UserInputPanel>
  <com.izforge.izpack.panels.install.InstallPanel
    id="install" />
  <com.izforge.izpack.panels.process.ProcessPanel
    id="process" />
  <com.izforge.izpack.panels.shortcut.ShortcutPanel
    id="shortcuts">
    <createShortcuts>false</createShortcuts>
  </com.izforge.izpack.panels.shortcut.ShortcutPanel>
  <com.izforge.izpack.panels.finish.FinishPanel
    id="finish" />
</AutomatedInstallation>
```

Example 3.1. sample auto-install.xml

3.6. Update Process

If Klaros-Testmanagement is already installed it is possible to update it to a newer version. The installer will update all files automatically. Further information of the installation process can be found in Section 3.3, "Installation Process". The Klaros-Testmanagement home folder remains untouched, so all settings, database connections and the content repository will be unaffected by the update process.

Important

Klaros-Testmanagement must be stopped before starting the update. Otherwise the installation process cannot finish successfully.

Note

It is highly recommended to create a backup of your database before starting the Klaros-Testmanagement update process. Most database products offer utilities for backing up database schemes, please refer to the relevant sections in the database vendor documentation. If you are using the pre-configured derby database you may skip this step but you should make sure to execute the following instructions.

In addition to the database Klaros-Testmanagement stores various data like configuration files, attachments and logs in the `.klaros` folder located in the home directory of the user running the Klaros-Testmanagement application server. In case of using the pre-configured derby database this includes the database as well. For a safe upgrade process you should create a backup of this folder as well before starting the update.

3.6.1. Incompatible Installation Detected

If you're trying to install a new version of Klaros-Testmanagement in some cases you will receive the message `Incompatible installation detected.` (see Figure 3.12, "Incompatible Installation detected"). In this case, Klaros-Testmanagement needs to be uninstalled first be-

fore a new version can be installed. As with the update process, all of your settings, database settings and the content repository will remain untouched by the uninstallation process.

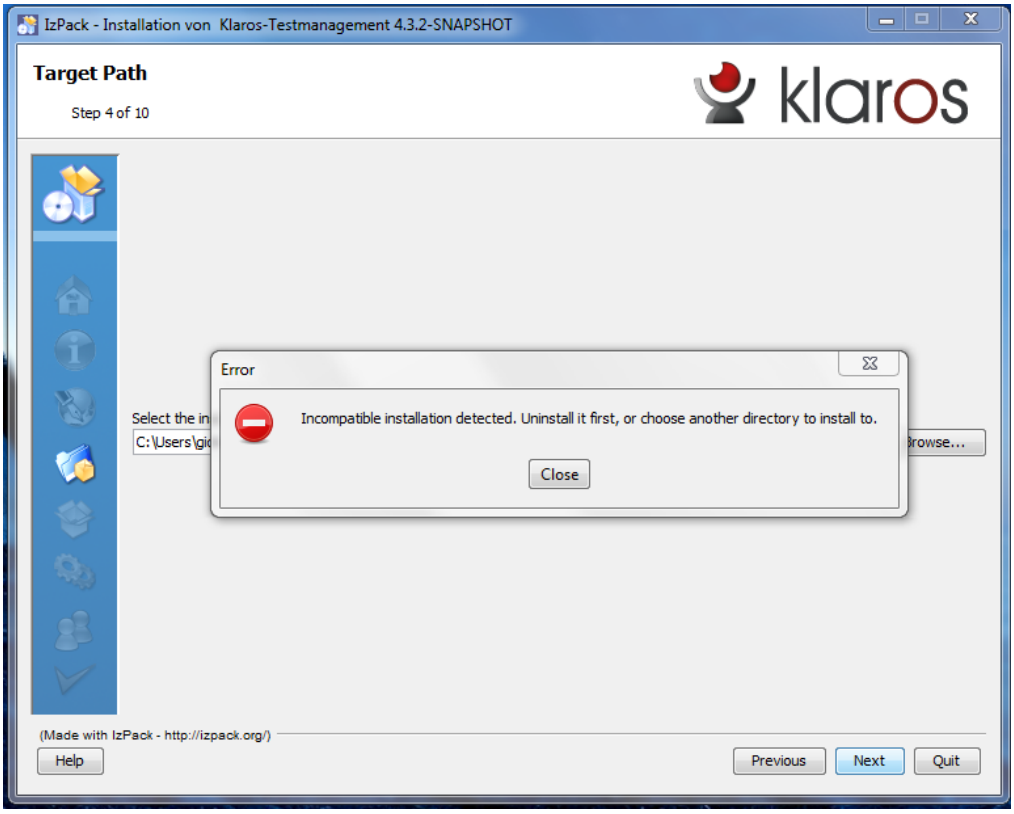


Figure 3.12. Incompatible Installation detected

When uninstalling Klaros-Testmanagement, the *Force the deletion of [Klaros-Testmanagement installation path]* checkbox needs to be set.

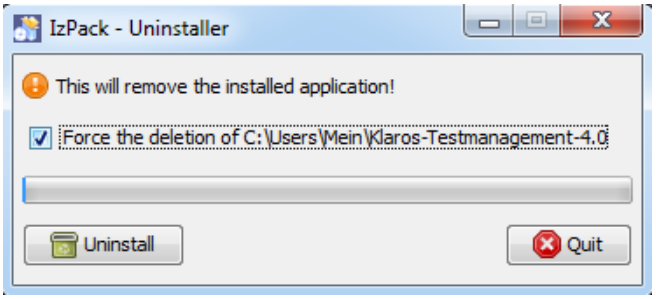


Figure 3.13. The Uninstaller

3.6.2. Manual Changes

Some settings may need manual adjustments when updating Klaros-Testmanagement from major revision 3 to 4, e.g. from 3.8.4 to 4.0.0.

3.6.2.1. Derby Dialect

If your installation of Klaros-Testmanagement uses the embedded Derby database, you should change the value of `hibernate.dialect` in `<userhome>/.klaros/hibernate.properties` to `org.hibernate.dialect.DerbyTenSevenDialect` for improved database handling.

3.6.2.2. Custom Reports

If you created custom reports (available with Klaros-Testmanagement Enterprise Edition), the pdf and excel namespace declaration in the XML root element needs to be adapted for each of them. Just replace the old URL

```
http://jboss.com/products/seam/pdf
```

given in an xmlns-attribute by

```
http://jboss.org/schema/seam/pdf
```

and

```
http://jboss.com/products/seam/excel
```

given in an xmlns-attribute by

```
http://jboss.org/schema/seam/excel
```

and press **Save**. For new reports the updated URL is already preset, so no further action is required.

If you miss to change the namespace, your document will not be rendered and you will receive the following error message:

```
Unexpected runtime exception: The rendered document has not been found in the document store
```

3.6.3. Database Migration

After updating Klaros-Testmanagement to a newer version it is possible that a database migration will be necessary. If so, Klaros-Testmanagement will show the following screen on startup. To start the migration an administrator account is needed. Before the migration process starts, the appropriate authentication domain should be selected (*LDAP* or *KLAROS*). Figure 3.14

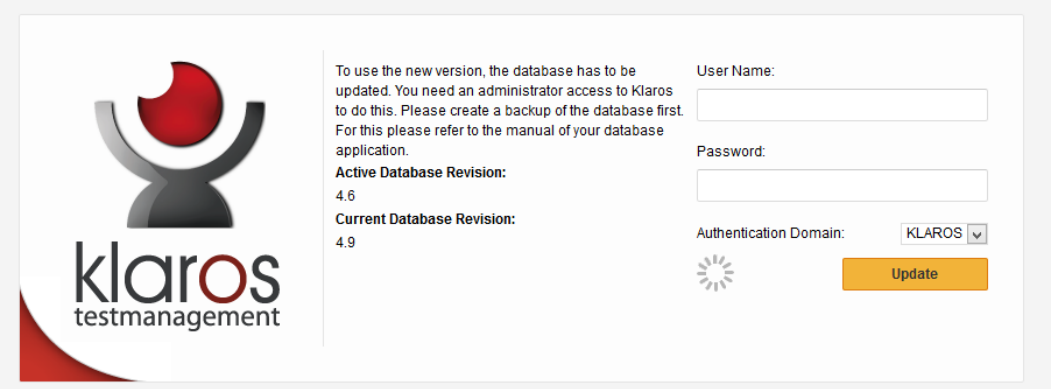

klaros
testmanagement
To use the new version, the database has to be updated. You need an administrator access to Klaros to do this. Please create a backup of the database first. For this please refer to the manual of your database application.
Active Database Revision:
4.6
Current Database Revision:
4.9
User Name:
Password:
Authentication Domain:
KLAROS
Update

Figure 3.14. Database Migration

3.6.3.1. Migration of installations prior version 3.0

In order to update Klaros-Testmanagement installations prior version 3.0 to version 3.0.0 or above it is mandatory to upgrade the installation to Klaros-Testmanagement 2.6.2 before attempting any further upgrade.

3.7. Important File Locations

There are two important locations in a Klaros-Testmanagement installation. One of these is the **installation folder** where application server and the klaros web application will be installed in. The other is the **home** folder containing runtime data like configuration files, search indices and attachments. Per default is the `.klaros` folder in the home folder of the user running Klaros-Testmanagement.

To allow a simple upgrade the installation/upgrade process will only change files in the installation folder and leaves the content of the home folder untouched.

On Microsoft Windows systems, by default the home folder is located at `C:\Documents and Settings\<username>\.klaros` for Microsoft Windows XP and earlier or `C:\Users\<username>\.klaros` for Microsoft Windows Vista, 7 and 8. This can also be opened by entering `%UserProfile%\.klaros` in an explorer window address bar or by entering `cd %UserProfile%\.klaros` on the command line.

On Linux systems this folder can be found using the path `~/.klaros` and is generally located at `/home/<username>/.klaros`.

On Mac OS X systems this folder is located at `/Users/<username>/.klaros` and can be found using `~/.klaros`.

Relocating the Klaros-Testmanagement home folder

By setting the environment variable `KLAROS-HOME` before starting the application the location of the home folder can be moved to another folder specified herein.

```
export KLAROS_HOME=/var/application-data/klaros
```

Heads up when moving the home folder!

If the application does not find a previous installation in the home folder it will bootstrap an empty default installation including a new local database. If this happens by accident, e.g. by providing a wrong home folder location it appears to the user that all previous data is gone.

Of course this is not the case. To resolve this situation point the KLAROS-HOME variable to the correct location and restart.

3.7.1. Log Files

The log files for Klaros-Testmanagement can be found in the `logs` folder in the installation directory and also in condensed form the `logs` folder within the `.klaros` directory.

Attaching the log files from these locations to a bug report helps the support team to quickly identify and fix any problems you encounter.

3.7.2. The Hibernate Properties File

The Hibernate properties file is located at `.klaros/hibernate.properties`. This file is used to specify the database to use for Klaros-Testmanagement. It contains the database type, URI and the authentication information. More information about changing the database can be found in Section 3.9, "Changing the Default Database".

3.7.3. Language Files

Klaros-Testmanagement may be customized to add new languages. This process is described in Section 4.1.1, "Defining Language Files". The file used to store the *Quote of the Day* entries is located at `.klaros/resources/quotes.txt`.

3.7.4. The Quotes File

The file used to store the *Quote of the Day* entries is located at `.klaros/resources/quotes.txt`.

3.7.5. The Derby Database

If you are using the default Derby database, this is located in the `.klaros/KlarosDerby` folder.

3.7.6. The Attachment Repository

All attachments are saved in the `.klaros/repository` folder. If the Klaros-Testmanagement installation and database are migrated to a different location this folder should be moved with them.

3.8. Running Klaros-Testmanagement

To access Klaros-Testmanagement, open the following URL in a web browser: `http://${host}:${port}/klaros-web/` where `${host}` is the IP address or domain name of the application server and `${port}` is the HTTP port defined in Section 3.3.6, "Step 6: User Data".

The first time Klaros-Testmanagement is executed, it will create a *.klaros* folder containing Klaros-Testmanagement and database property files. The *.klaros* folder is created in the home directory of the current user. For Microsoft Windows XP users it is located in *C:\Documents and Settings\username*, for Microsoft Windows Vista and Microsoft Windows 7/8 users it is in *C:\Users\username*, and for Linux users in */home/username* by default.

3.9. Changing the Default Database

By default Klaros-Testmanagement uses the Apache Derby Database which requires no further installation or maintenance but is lacking performance in comparison to other implementations. For production systems, it is strongly recommended to use a full-blown database system, such as the open source PostgreSQL or MariaDB/MySQL database servers or the commercial Microsoft SQLServer or Oracle database products.

To switch to another database system, Klaros-Testmanagement must be stopped, and the file `hibernate.properties` located in `<user.home>/.klaros/hibernate.properties` must be edited as shown below.

The `hibernate.connection.url` property must match the location of the database in your network. Please consult your database administrator for the `hibernate.connection.username`/ `hibernate.connection.password` credentials to use.

To use a Microsoft SQL-Server change the content of the file to:

```
hibernate.dialect=org.hibernate.dialect.SQLServer2008Dialect
hibernate.connection.driver_class=com.microsoft.sqlserver.jdbc.SQLServerDriver
hibernate.connection.url=jdbc:sqlserver://localhost:1433;databaseName=KLAROS
hibernate.connection.username=root
hibernate.connection.password=root
```

For MariaDB/MySQL change the content of the file to:

```
hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.connection.url=jdbc:mysql://localhost:3306/klaros
hibernate.connection.username=root
hibernate.connection.password=root
```

For Oracle please use the following:

```
hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
hibernate.connection.driver_class=oracle.jdbc.driver.OracleDriver
hibernate.connection.url=jdbc:oracle:thin:@localhost:1521:klaros
hibernate.connection.username=root
hibernate.connection.password=root
```

To use PostgreSQL change the content of the file to:

```
hibernate.connection.driver_class = org.postgresql.Driver
hibernate.connection.url = jdbc:postgresql://localhost/klaros
hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
hibernate.connection.username=root
hibernate.connection.password=root
```

An exhaustive list of all parameters can be found in the Hibernate Core Manual.

Database Instance Creation

Klaros-Testmanagement will not automatically create either the database instance (`klaros` in the above example) or the database user (user `root` with password `root` in the above example) in the database server. Creating a database instance and adding a user is described in the corresponding database manual and will not be covered here. The database user needs permissions to create, drop and alter tables to properly bootstrap the Klaros-Testmanagement database instance.

Make sure to activate support for UTF-8 character sets

Depending on your database product it may be needed to manually activate support for UTF-8 character sets, which may be needed to support languages with uncommon character sets. Especially MySQL is known for coming with a limited character support in its initial configuration. The following example shows how to activate this by editing the `mysql.cnf` configuration file.

```
character-set-server=utf8
collation-server=utf8_general_ci
```

Example 3.2. Additional entries in mysql.cnf to support UTF-8 character sets

3.10. Installing Klaros-Testmanagement as a Service

For day-to-day usage of Klaros-Testmanagement, it is necessary that the application is available after a reboot of the machine it is installed on. Depending on the operating system, different measures have to be taken to achieve this.

3.10.1. Installing Klaros-Testmanagement as a Linux service

To start Klaros-Testmanagement automatically the startup script shown below should be saved to the **`/etc/init.d`** directory as **`klaros`** and selected for the appropriate run level. Please refer to the documentation of your linux distribution for details.

The values of the `KLAROS_USER` and `KLAROS_PATH` fields must first be replaced with the intended user id and installation path. It is important that the script is executable. This can be ensured with the command **chmod a+x klaros**.

```
#!/bin/bash
#
#
# This will be ignored by systems that don't use chkconfig.
# chkconfig: 345 98 2
# description: klaros (inside Tomcat, a servlet container)
#
KLAROS_USER=klarosdemo
KLAROS_PATH="/opt/klaros-testmanagement"
/bin/su - "${KLAROS_USER}" -c "${KLAROS_PATH}/bin/catalina.sh $@ >/var/tmp/klaros.log 2>&1"
```

3.10.2. Installing Klaros-Testmanagement as a Windows service

Under the Windows operating system family the supplied **service.bat** script can be used to install the Tomcat application server as a Windows service.

Please refer to the official Apache Tomcat 8 Windows service HOW-TO for further details.

Change the Service User Account!

Per default the created Windows service will use the `LocalSystem` account to run the Klaros-Testmanagement application. This setting must be changed to the account of the user you intend to use for running Klaros-Testmanagement in the service account settings. This change is needed due to the fact that the default location for the `.klaros` folder in the user account home.

Skipping this vital step will lead to a newly created `.klaros` home folder in the `C:\Windows\System32\systemuser` folder (this location may vary with your Windows version in use; newer version may use `C:\Windows\system32\config\systemprofile` instead) containing a new empty database, which will now be used by the service instead of the database present when starting Tomcat manually.

If you encounter this situation, then shut down the service, delete the `.klaros` folder in the `Windows\System32` folder, change the user account settings of the service and finally restart the service.

Make sure to use a Java architecture that matches with your Tomcat installation

Upon first installation of Klaros-Testmanagement the installer chooses a matching Tomcat Windows installation (32-Bit vs. 64-Bit) for the Java runtime architecture it was started with.

If you want to change the desired Java Runtime Architecture later (e.g. switching from 32-Bit Java to 64-Bit Java) please make sure that you re-install your distribution with the desired Java runtime in your system path. If in doubt check by running **javaw -version** from the **cmd** windows command line processor.

The service will refuse to start if you choose the wrong Java architecture. A typical error output for running a 32-Bit Tomcat installation in conjunction with a 64-Bit Java looks like the following:

```
Commons Daemon procrun (1.0.15.0 32-bit) started
Debugging 'tomcat7' service...
Starting service...
%1 is not a valid Win32 application.
Failed creating java C:\Program Files\Java\jre1.8.0_45\bin\server\jvm.dll
%1 is not a valid Win32 application.
ServiceStart returned 1
%1 is not a valid Win32 application.
Debug service finished with exit code 1
Commons Daemon procrun failed with exit value: 3 (Failed to run service as console application)
%1 is not a valid Win32 application.
```

You may update the Java virtual machine which the service uses in the `Java` tab of the **tomcat8w** administration interface.

Adjust the memory settings when updating older installations!

Klaros-Testmanagement needs a larger amount of memory than the default Java installation suggests. If you are using a version of Klaros-Testmanagement prior to release 3.8.0 it is necessary to supply the following settings to the Tomcat configuration which can be set in the **tomcat8w** administration interface by adding the line `-XX:MaxPermSize=128m` to the `Java Options` text area in the `Java` tab as shown below. In addition, a reasonable amount of maximum memory should be defined here as well. Later versions to not require this as the installer with allow to change these settings.

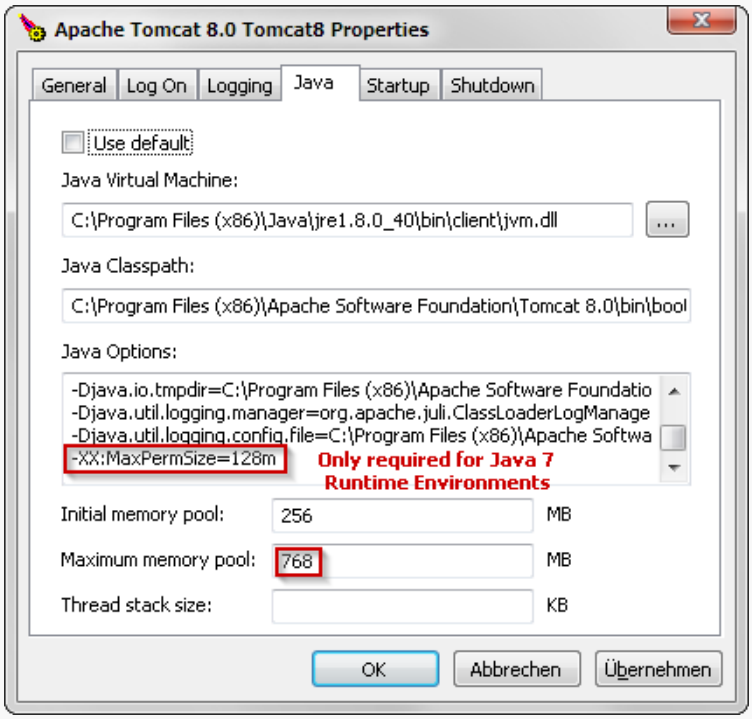


Figure 3.15. Additional Tomcat memory settings

3.11. Monitoring Klaros-Testmanagement

Using the built-in JMX Support in Tomcat it is possible to monitor the Klaros-Testmanagement installation using any JMX capable Monitoring tool (like e.g. VisualVM).

Add the following lines to bin/catalina.sh:

```
# Set JMX connector
RMI_SERVER='[FQDN]'

JMXREMOTE="-Dcom.sun.management.jmxremote"
JMXREMOTE="$JMXREMOTE -Dcom.sun.management.jmxremote.port=[PORT]"
JMXREMOTE="$JMXREMOTE -Dcom.sun.management.jmxremote.ssl=false"
JMXREMOTE="$JMXREMOTE -Dcom.sun.management.jmxremote.authenticate=false"
JMXREMOTE="$JMXREMOTE -Dcom.sun.management.jmxremote.ssl=false"
```

```
JAVA_OPTS="$JAVA_OPTS $JMXREMOTE -Djava.rmi.server.hostname=$RMI_SERVER"
```

Where [FQDN] and [PORT] denote the fully qualified domain name and port of the remote JMX server you want to deliver messages to.

3.12. Configuring External Issue Management Systems

Depending on the type of Issue Management System, some configuration is needed to connect it to Klaros-Testmanagement. This section describes the needed installation steps for each system.

Name	Version
Bugzilla	3.0 or later
JIRA	3.7 or later
Mantis	1.1.8 or later.
Redmine	1.x and up. Users of previous Redmine versions are strongly encouraged to upgrade their Redmine installations and their WS-API plugins.
Trac	0.10 or later

Table 3.3. Supported External Issue Management Systems

3.12.1. JIRA Configuration

To allow the connection of Klaros-Testmanagement to the JIRA server instance the JIRA RPC Plugin has to be activated in your JIRA installation. You find this option as *Accept remote API calls* in **General Configuration** under **Global Settings**. Then you need to enable the JIRA RPC Plugin in **Plugins** under **System** in the left-hand menu.

3.12.2. Redmine Configuration

3.12.2.1. Redmine1.x

The Klaros-Testmanagement Redmine Integration for Redmine 1.x requires the installation of the Redmine-Mylyn-Connector 2.7.6 for Redmine 1.0 - 1.4.

Prerequisites

1. Redmine 1.0.x.STABLE with enabled RESTFUL-API

Procedure 3.1. Installation

1. Open a shell in your Redmine directory
2. Execute

```
ruby script/plugin install --force \
```

```
git://redmin-mylyncon.git.sourceforge.net/gitroot/redmin-mylyncon/redmine-mylyn-connector
```

Note

These steps require the ruby-git bindings

3.12.2.2. Redmine 2.0 and later

The Klaros-Testmanagement Redmine integration for Redmine 2.0 and later requires the installation of the Redmine-Mylyn-Connector 2.8.2 or later. Please follow the installation notes which can be found at http://danmunn.github.com/redmine_mylyn_connector/.

3.12.2.3. Redmine 3.0 and later

The Klaros-Testmanagement Redmine Integration for Redmine 3.0 and later requires the installation of a patched version of Redmine-Mylyn-Connector 2.8.2. Please follow the installation notes which can be found at http://danmunn.github.com/redmine_mylyn_connector/. but use **git://github.com/joaopedrotaveira/redmine_mylyn_connector.git** instead of the documented **git://github.com/danmunn/redmine_mylyn_connector.git** command.

3.12.2.4. Redmine common configuration

Figure 3.16, "Redmine Plugin Section" shows the Redmine Plugin section after the successful installation of the plugin.

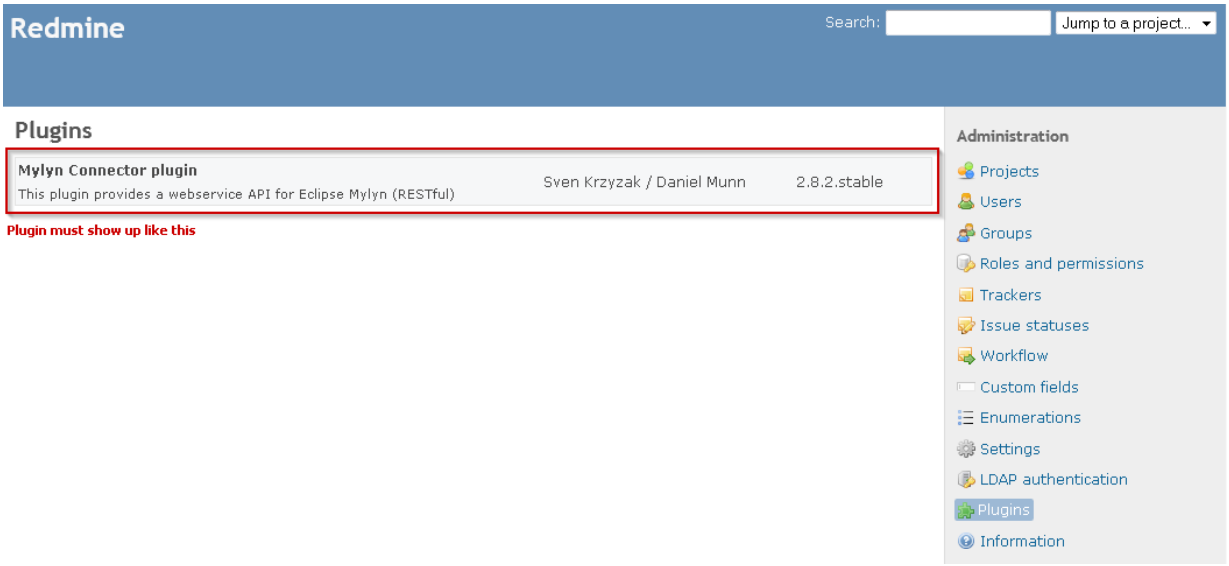


Figure 3.16. Redmine Plugin Section

Note

In order to use Klaros-Testmanagement Redmine integration, the *Authentication Required* and the *Enable REST Web Service* checkbox in the Authentication section of the Redmine settings must be checked (Figure 3.17, "Redmine Authentication Section").

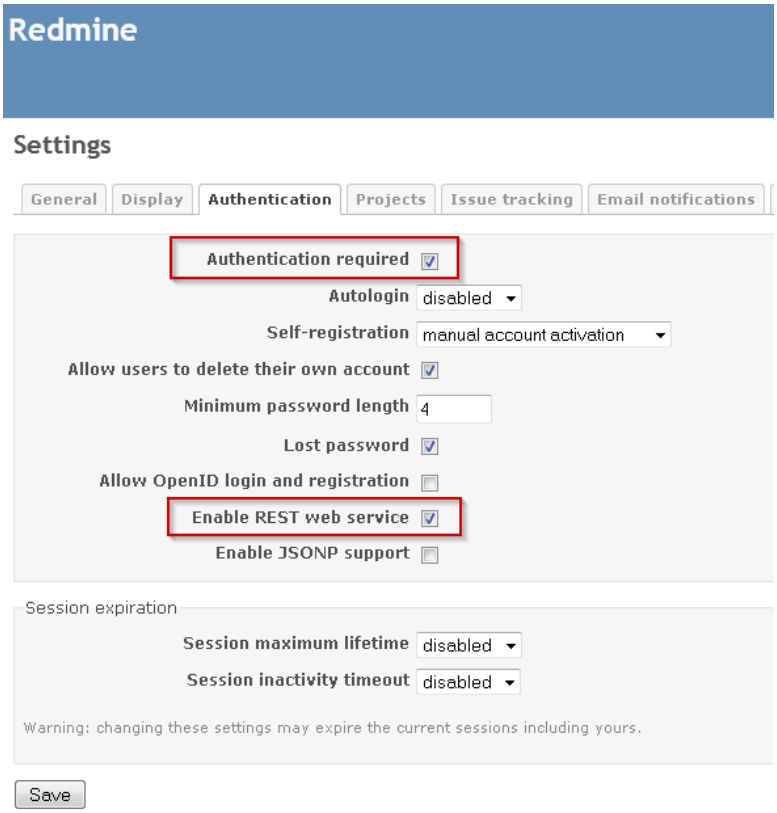


Figure 3.17. Redmine Authentication Section

The following combinations have been successfully tested:

- Redmine 1.0.4 / Redmine Redmine-Mylyn-Connector/WS-API Plugin 2.7

  Redmine 1.2 / Redmine Redmine-Mylyn-Connector/WS-API Plugin 2.7.4.stable

  Redmine 2.1 / Redmine Redmine-Mylyn-Connector/WS-API Plugin 2.8.2.stable

  Redmine 2.3 / Redmine Redmine-Mylyn-Connector/WS-API Plugin 2.8.2.stable

  Redmine 2.5 / Redmine Redmine-Mylyn-Connector/WS-API Plugin 2.8.2.stable

  Redmine 3.0 / Redmine Redmine-Mylyn-Connector/WS-API Plugin 2.8.2.master

Later versions are expected to work but are currently not tested.

3.12.3. Trac Server Configuration

The Klaros-Testmanagement Trac integration requires the installation of the TracXMLRPC Plugin. The download archives and installation notes are available at http://trac-hacks.org/wiki/XmlRpcPlugin/. After successful installation of the plugin, it has to be activated using the Trac web administration interface. Figure 3.18, “Trac Plugin Section” shows the Trac Plugin section after the successful installation of the plugin.

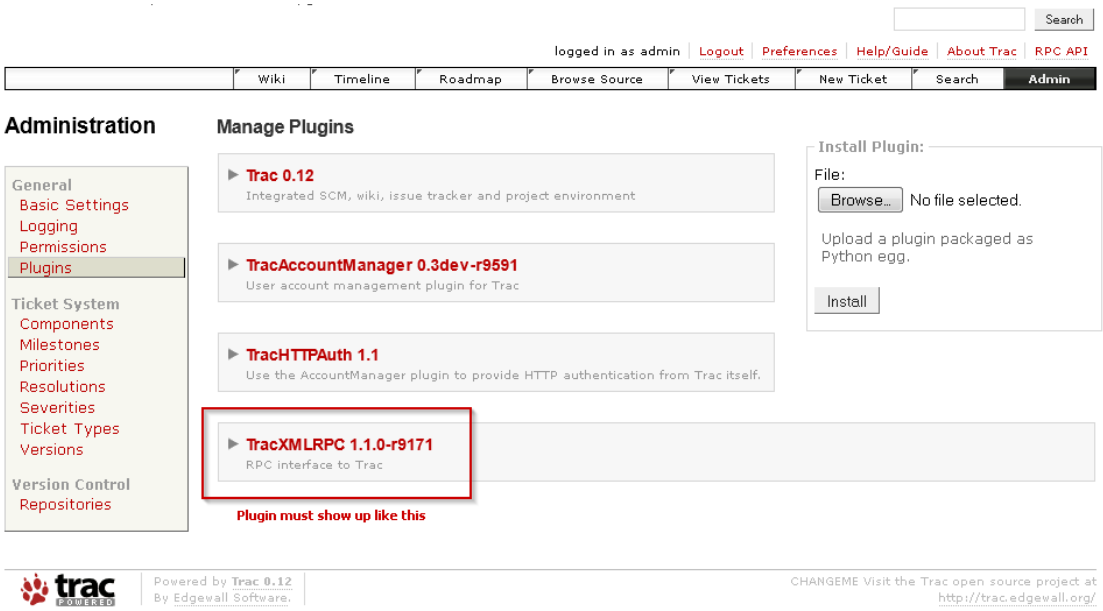


Figure 3.18. Trac Plugin Section

To make the plugin accessible, authenticated users must be able to access the XML_RPC plugin as shown below.

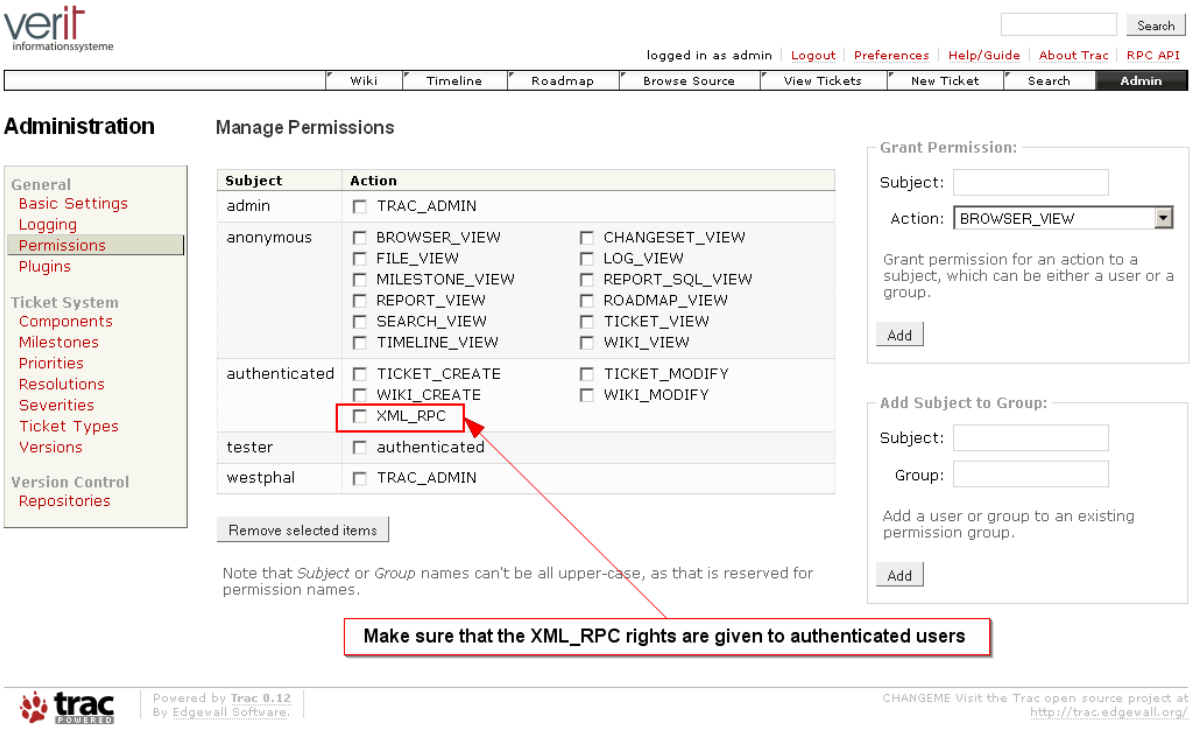


Figure 3.19. Trac Permissions Section

The following combinations have been successfully tested:

- Trac 0.12 / TracXMLRPC 1.1.0

Later versions are expected to work but currently not tested.

3.13. SSL Support

Note on self-signed certificates: If you are accessing HTTPS servers with self-signed certificates, you will need to import those certificates into your Java trusted keystore. One instruction on how to achieve this can be found at http://stackoverflow.com/questions/2893819/telling-java-to-accept-self-signed-ssl-certificate.

3.14. Uninstall

To uninstall Klaros-Testmanagement select *Uninstall Klaros-Testmanagement Server* from the start menu (if running under Windows) or just delete the installation folder manually.

If you have added an installation as a service as described in Section 3.10, “Installing Klaros-Testmanagement as a Service” please remember to deactive the Windows service or Linux init script.

Home Folder

Just like during the upgrade process uninstalling Klaros-Testmanagement will not touch your home folder (`${user.home}/.klaros`).

So if you uninstalled by accident re-installing will let you restore your system safely, else remove feel free to remove that folder manually.

Chapter 4. Customization

Some parts of Klaros-Testmanagement may be customized by the user, including the display languages and the quote of the day file.

4.1. Languages

Klaros-Testmanagement supports the addition of custom languages. To add a language to Klaros-Testmanagement, a language file must first be added to the `.klaros/resources/messages` folder in the home directory of the user running Klaros-Testmanagement.

The available languages can be enabled/disabled using the *Languages* page in the system parameters section. The process of enabling and disabling available languages is described in Section 10.5.4.1, “Enabling and Disabling Languages”.

4.1.1. Defining Language Files

Adding a valid language file to the `.klaros/resources/messages` directory causes the language to become available in Klaros-Testmanagement. A valid language file has a name of the form `messages_${language}_${COUNTRY}_${variation}.properties` where `${language}` is the two-letter ISO 639 code for the language in lower-case, `${COUNTRY}` is the two-letter ISO 3166 code for the country in upper-case and `${variation}` is an optional, user-definable code for the variation of the language (e.g. dialect).

Language File Encoding

The language file must use ISO-8859-1 encoding with Unicode escapes (`\uXXXX`).

If you prefer editing these files as UTF-8 please use a converter like **native2ascii** or **iconv** to translate them to a proper format.

After Klaros-Testmanagement is run for the first time, the American English, German and fallback language files will be copied to the `.klaros/resources/messages` folder. In order to create a new language file, the text should be copied out of one of these files into an appropriately named file. The contents of the file consist of key value pairs in the form: `key=This sentence is the value`. None of the keys should be altered. The values (the text after the '=') should be translated into the target language.

When Klaros-Testmanagement is started, the default language files (`messages.properties`, `messages_en_US.properties` and `messages_de_DE.properties`) are copied to the `.klaros/resources/messages` folder, overwriting any changes made to these files. Therefore, it is advised that any user changes to the American English and German interfaces be made in a file with the variation in the filename set, e.g. `messages_en_US_companyName.properties` in order to persist changes.

As well as copying over the default files, Klaros-Testmanagement also merges any changes with user-defined files. If key-value pairs are added to the default files during an update, these will also be added to the end of any user-defined language files, along with a comment stating when the entries were added. These entries should be translated by the creator of the custom language file.

4.2. Custom Quotes

Klaros-Testmanagement allows the use of a custom quotes file. This allows the customization of the *Quote of the Day* field shown on the login screen. A custom quotes file can be uploaded to Klaros-Testmanagement using the form on the *General Settings* page can be placed directly in the `.klaros/resources` folder in the home directory of the user running Klaros-Testmanagement.

A custom quotes of the day file is simply a text document with the extension `.txt` which uses ISO-8859-1 encoding with Unicode escapes and contains one quote per line. The quote displayed changes each calendar day. The process of uploading a custom quotes file is described in Section 10.5, “System”.

Chapter 5. Functional Overview

This chapter gives a coarse overview of the usage of the application.

Warning

The browser back button is not supported by Klaros-Testmanagement. The behavior of the web application is unpredictable when it is used.

5.1. Login

Figure 5.1 shows the login screen of Klaros-Testmanagement. To login just enter the user name in the *User name* field and the associated password into the *Password* field.

Figure 5.1. Login Screen

The Klaros-Testmanagement database predefines three default users accounts with following roles:

- Administrator

  Username: `admin` / Password: `admin`

- Manager

  Username: `manager` / Password: `manager`

- Tester

  Username: `tester` / Password: `tester`

For a description of the users roles and the permissions associated with each role, see Appendix A, *Role Permission Overview*.

Confirm the login to Klaros-Testmanagement by clicking the **Login** button.

Klaros-Testmanagement is multilingual

By default Klaros-Testmanagement chooses the language that is determined by the locale settings of the server Klaros-Testmanagement is installed on. If you want to choose a different language you may select the corresponding flag at the bottom right corner of the login screen. The language of the user interface changes immediately and the choice will be stored automatically.

After a successful login the *Maintain Project* page is shown (Figure 5.2).

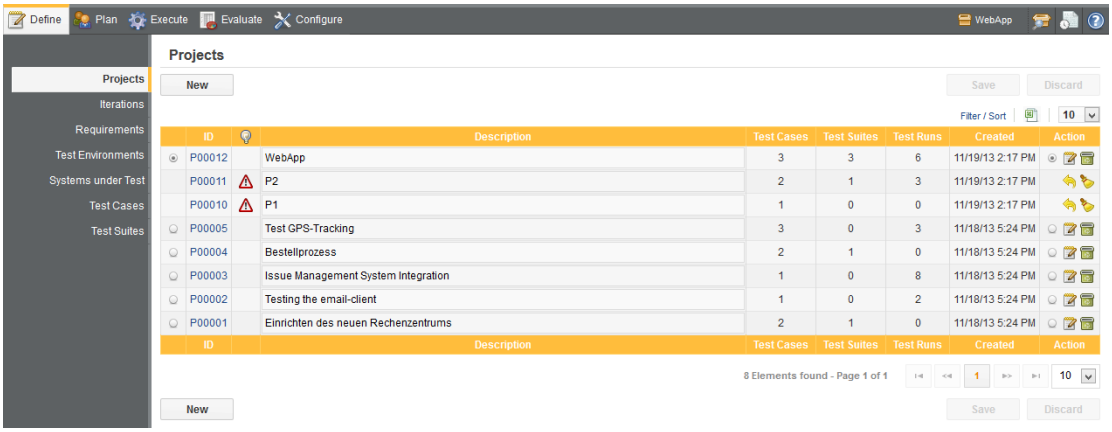


Figure 5.2. Maintain Projects Page

Klaros-Testmanagement uses a horizontal menu on the top of the screen. These categories resemble activities in the testing project. The function categories presented in this menu are:

Define

In the *Define* section of Klaros-Testmanagement all artifacts that are needed in a project can be defined and viewed. These artifacts are the project itself, the test environments, the systems under test, the test cases and the test suites. For the Klaros-Testmanagement Enterprise Edition and Klaros-Testmanagement Mobile Edition this list also includes project iterations and requirements.

Plan

The *Plan* section of Klaros-Testmanagement is used to define and maintain jobs, e.g. the execution of tests which can be assigned to individual users and tracked accordingly. This section is only available in Klaros-Testmanagement Enterprise Edition and Klaros-Testmanagement Mobile Edition.

Execute

The *Execute* section of Klaros-Testmanagement allows execution of jobs, manual test cases and manual test suites. Before a test is executed, the tester must select a combination of system under test and test environment for which he executes the test. He is then guided step by step through the manual test. Comments can be added at each step.

	If the tester is interrupted during the execution of a long test suite, he can stop execution at any time and continue the test suite execution later.
Evaluate	The *Evaluate* section provides all functionality for analyzing the project. It provides coarse overview reports in the dashboard, as well as fine-grained information, so that each execution of a test can be traced and visualized.
Configure	The *Configure* section contains access to administrative tasks, e.g. the system settings, user management or the import and export of projects.

The following chapters will describe each category in detail.

5.2. Main Functions

This section describes the use of some common features of Klaros-Testmanagement.

5.2.1. The Help Menu

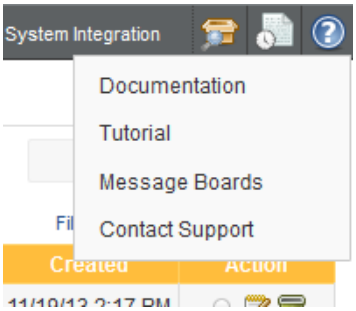


Figure 5.3. The Help Menu

The help menu provides quick links to various Klaros-Testmanagement resources. Pressing the help menu icon opens a menu appears displaying the following links:

The Help Menu

Documentation	Opens up the section of the Klaros-Testmanagement documentation corresponding to the currently displayed page.
Tutorial	Opens up the Klaros-Testmanagement tutorial.
Message Boards	Opens up the Klaros-Testmanagement message boards.
Contact Support	Prepares an e-mail to be sent to the Klaros-Testmanagement support team. This option should be used when experiencing an issue using Klaros-Testmanagement which can't be solved using the documentation or the message boards.

5.2.2. The User Menu

Figure 5.4. The Help Menu Button

The user menu button on the upper right of Klaros-Testmanagement displays the full name of the user that is currently logged in.

Figure 5.5. The Help Menu

Pressing the user menu button opens up the user menu, where the user can change the language and log out of Klaros-Testmanagement.

5.2.3. Search / Quick-Select

Depending on the Klaros-Testmanagement edition, the upper right corner (see Figure 5.6) either contains a search field for fast navigation to elements with a given ID or for a fully featured full-text search in the current selected project. These are described in the following sections.

Figure 5.6. The Search Field

5.2.3.1. Full Text Search

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

The *Search* field allows the user to search for object ids or words or phrases contained in fields of objects. This search function uses Apache Lucene, which provides a powerful syntax for searching individual fields or with wildcards. This syntax is described on the Apache Lucene website.

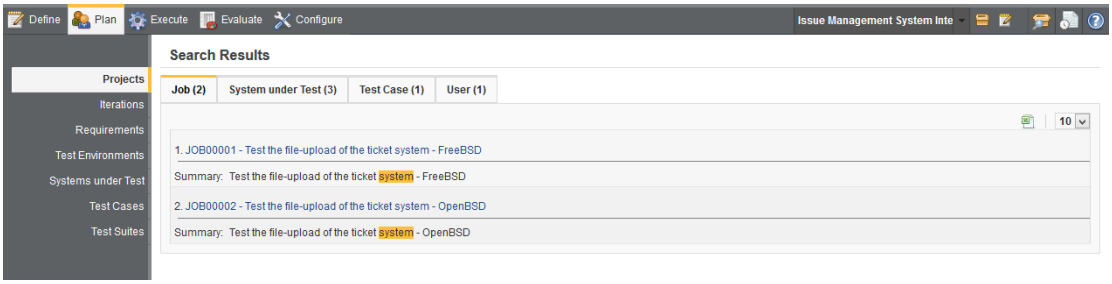


Figure 5.7. The Search Results Page

Figure 5.7 shows the *Search Results* page. This page displays each object containing the search term, and highlights the occurrences.

Figure 5.8 shows the message which appears in the log panel upon completion of a search.

Figure 5.8. The Status Bar Search Report

5.2.3.2. Quick-Select

For known element ids (like e.g. TC00001) a quick navigation field is present in the Klaros-Testmanagement Community Edition. It currently supports the following element types:

- Project
- Test Environment
- System under Test
- Test Case
- Test Suite
- Test Case Result
- Test Suite Result
- Test Run
- Issue Management System

Please note, that ids will usually be searched for in the current project only.

Abbrevation Support

It is possible to use case independent abbreviated forms of IDs, eg. `TC00001`, `TC1` or `tc1` will all resolve to the same element.

5.2.4. The Log Panel

The current selected project is: Test Development (P00016)
INFORMATION 10:32:04 The current selected project is: Test Development (P00016)
INFORMATION 10:32:03 The Project 'P00016' has been successfully created
INFORMATION 10:31:35 The current selected project is: Test GPS-Tracking (P00005)
INFORMATION 10:31:35 The current selected iteration is: Sprint 1 (ITR00001)
INFORMATION 10:31:33 The current selected project is: WebApp (P00012)

Figure 5.9. The Log Panel

The log panel shows status messages (like warnings or information) to the user. Initially, only the latest status message is displayed. By clicking on the **+** icon on the right side of the log panel, all status messages since the last login are displayed. Clicking the **x** icon closes the log panel. It can be opened again at any time by clicking on the log panel icon in the navigation bar.

Figure 5.10. The Log Panel Icon

5.2.5. Filtering and Sorting

It is possible to filter and sort the displayed objects on most pages in Klaros-Testmanagement. The *Filter / Sort* link above the table opens the panel, which contains two tables. The left-hand table contains the active filters, and the right-hand table contains the active sort orders.

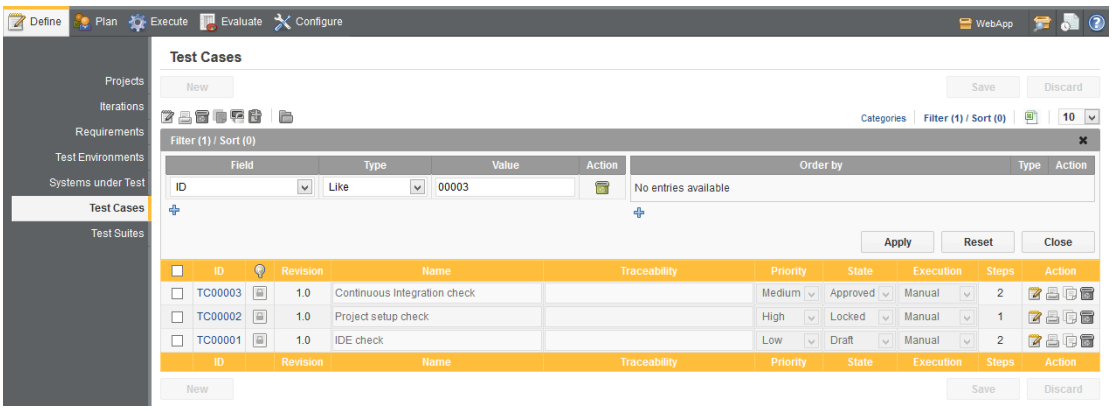


Figure 5.11. The Filter and Sort Panel

Below the two tables are two buttons:

Apply — The **Apply** button applies the filter and sorting criteria specified in the two tables and presents the filtered and sorted results.

Reset — The **Reset** button clears the filter and sort tables and displays all objects.

5.2.5.1. Specifying Filtering Criteria

Each line in the filter table represents a criterion which restricts the result set.

Clicking the icon adds a new, empty row for a filter criterion to the table. The filtering criteria table has four columns:

- The *Field* column denotes the field in the table that the criterion filters for. The fields which can be filtered in the *Filter / Sort* panel vary from page to page, but generally include the main attributes of the objects which are displayed, including user defined properties if available.
- The *Type* column denotes the operator that is used for the criterion. The following operators are available in Klaros-Testmanagement:

Equals	Only the items that exactly match the value of the criterion are shown.
Not equals	The items that do not equal the value of the criterion are shown in the result table.
Greater than or equals	Items that are greater than or equal the value of the criterion are shown.
Less than or equals	Items that are less than or equal the value of the criterion are shown.
Greater than	Items that are greater than the value of the criterion are shown.
Less than	Items that are less than the value of the criterion are shown.
Like	Items that have a part that matches value of the criterion are shown.

- The *Value* column defines the value for the filtering criterion.
- The *Action* column contains a button ( ) for deleting the criterion on the current line.

5.2.5.2. Specifying Sorting Criteria

The objects displayed in Klaros-Testmanagement can be sorted. The criteria for the sorting are specified in the right-hand table in the *Filter / Sort* panel. It is possible to specify more than one sorting option.

If there is more than one sorting option, the uppermost row in the sorting criterion table has the highest priority and the lowest row has the lowest priority.

When the icon is clicked, a new empty row for a sorting criterion is appended at the bottom of the table. Each row in the sorting options table has a delete button ( ) in the *Action* column. Pressing this button removes the sorting criterion.

The sorting options table contains three columns:

- The *Order By* column denotes the field in the results table that the sorting applies to. The fields that can be sorted vary from page to page, but generally include most attributes of the objects being displayed.

- The *Type* column defines the direction of the sorting sequence, i.e. descending or ascending.
- The *Action* column contains a button for deleting the sorting criterion.

5.2.6. Print Pages

Feature only available in Klaros-Testmanagement Enterprise Edition

Most objects in Klaros-Testmanagement may be viewed on a printer-friendly page. This print page is viewed by clicking on the print icon . This icon appears in the *Action* column of the table, above tables with bulk options and as a large icon at the top right of *Details* pages.

The print page displays all of the fields of the object on a single page. The print pages for objects which contain other objects (e.g. test suites contain test cases) also contain print options at the top. These allow the user to limit the information which is printed, e.g. the *Details* option on the test suite print page limits the displayed test case information to the *ID* and *Name* of the test cases contained in the test suite. By selecting the *Suppress empty fields* checkbox, all fields which do not contain contents are suppressed from the print view. Figure 5.12 shows the print page for a test suite.

Note

The selected print options will preserved during a user session.

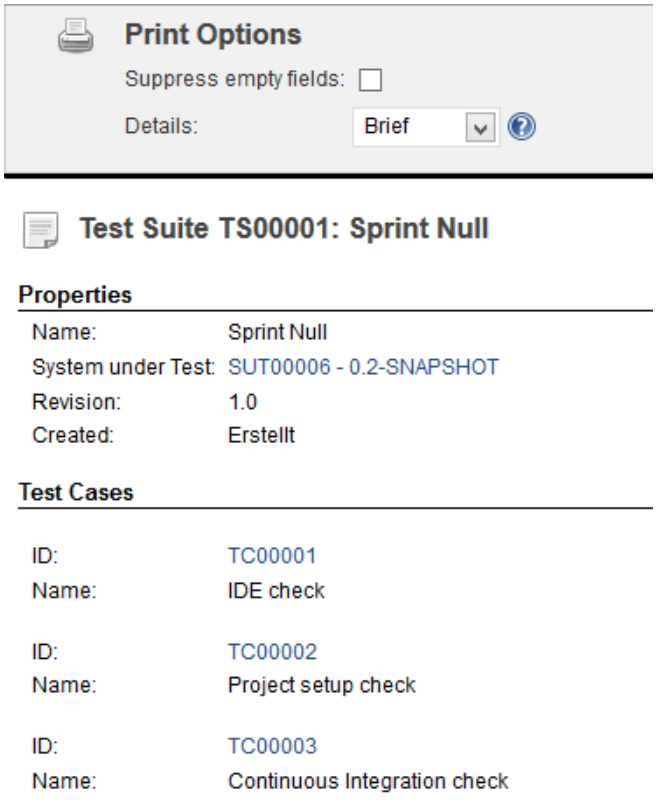


Figure 5.12. The Test Suite Print Page

5.2.7. Categorization

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

In order to manage a large numer of artifacts, it is possible to catigorize them using arbitrary cretiria. This feature can be used with *Iterations*, *Requirements*, *Systems under Test*, *Test Environments*, *Test Cases* and *Test Suites*. The categorization panel is opend by clicking the *Categories* link above the table in question.

In the categorization panel, users may create *Category Groups* and *Categories*. Objects may be assigned to one category in each group. There is no limit on the amount of categories or category groups which can be created.

In order to apply categorization, the *Categories* panel must be open. The Categories panel contains 3 different views, the *Edit* view, the *Tree* view and the *Menu* view. If no categories have been defined, only the edit view can be used. Each view allows selection of the active category group through the use of a dropdown menu, or selection of the blank category group to disable categorization.

5.2.7.1. The Edit View

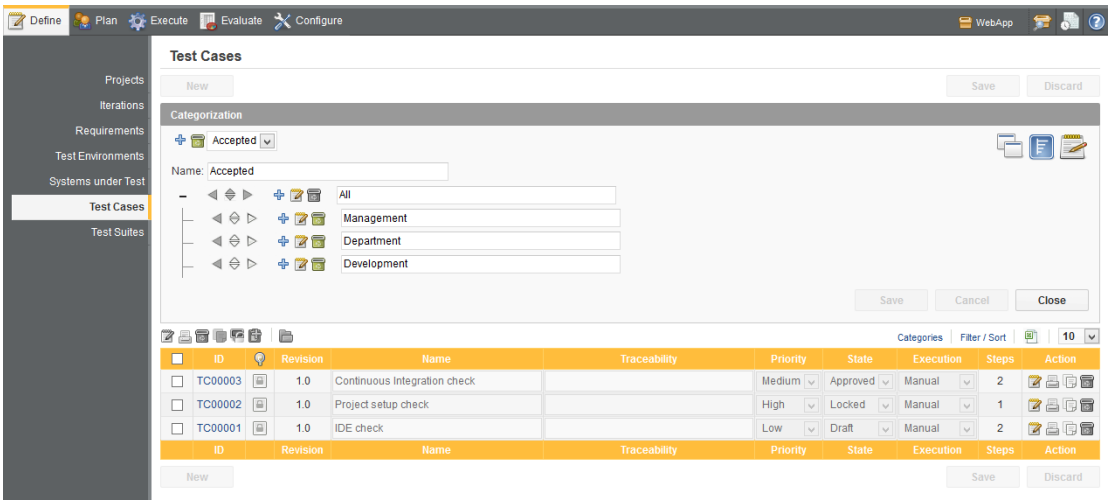


Figure 5.13. The Categorization Edit View

This view allows the creation and naming of category groups and categories, as well as the organization of category hierarchies. Pressing the add button at the top of this view creates a new category group, and enables the use of the category group dropdown menu. If a category group is selected, it can be renamed in this view.

Categories may also be added and edited in this view. Pressing the add button on a category adds a new sub-category. By default, a new category tree contains one category, and all future categories are added as sub-categories of this *Root Category*. Categories may be renamed by changing the value in the name field and also given a description through use of the edit button.

Changing the hierarchy of categories and ordering them is also possible in the editview through use of the arrow buttons.

5.2.7.2. The Tree View

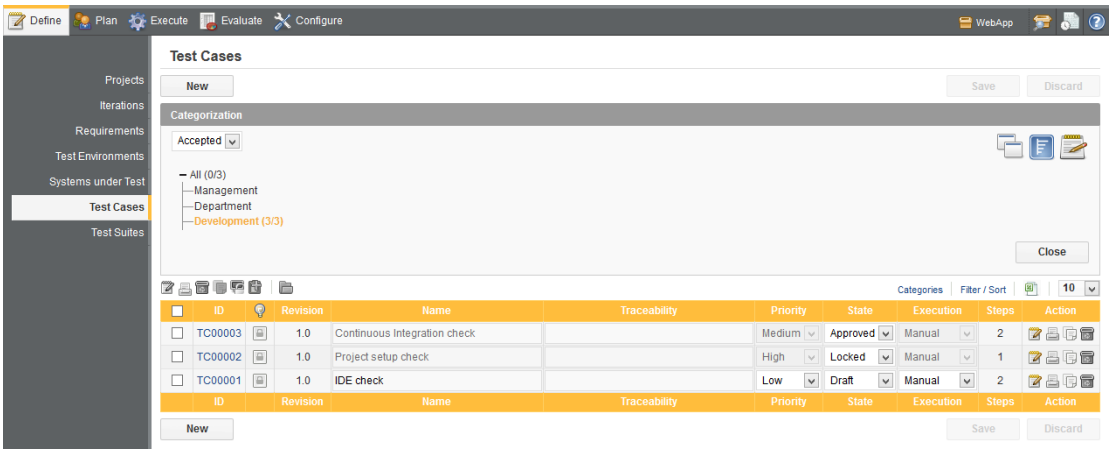


Figure 5.14. The Categorization Tree View

This view shows an overview of the category structure. The available categories are represented as a collapsible tree, with the name of each category is shown, along with the num-

ber of objects it contains. The description of each category is shown as a tooltip when its name is hovered over with the mouse pointer.

5.2.7.3. The Menu View

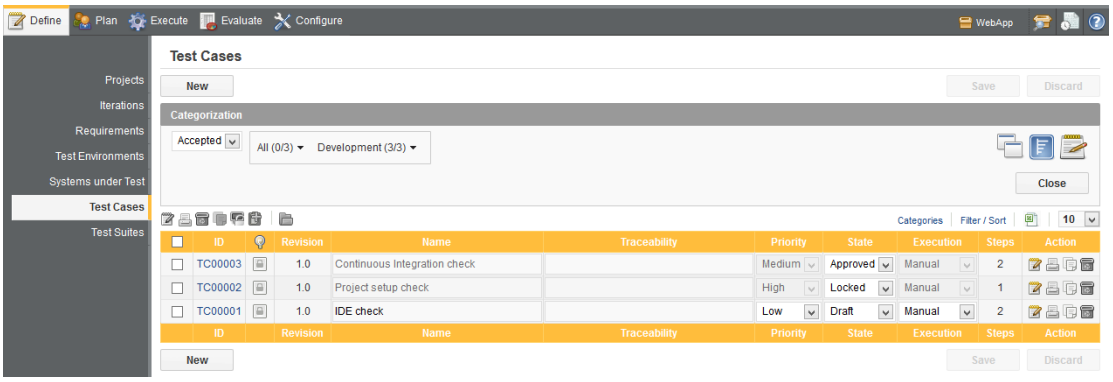


Figure 5.15. The Categorization Menu View

This view allows more compact use of the category panel. It contains a drop-down menu which enables the user to choose any of the available categories, through a submenu system.

5.2.8. Conflict Handling

Klaros-Testmanagement allows for conflict resolution in a multi-user environment.

When saving an object, Klaros-Testmanagement attempts to automatically resolve any possible conflicts with concurrent changes made by other users. Non-conflicting changes (i.e. any changes made by other users which do not affect fields which the current user has changed will be merged with no additional user input.

If there are any conflicting changes, i.e. changes to fields which were also changed in the mean time by another user, the *Conflict Resolution Dialog* will be displayed to allow the user to choose how the conflicts should be resolved.

Figure 5.16 shows the *Conflict Resolution Dialog* for a conflicting test case change.

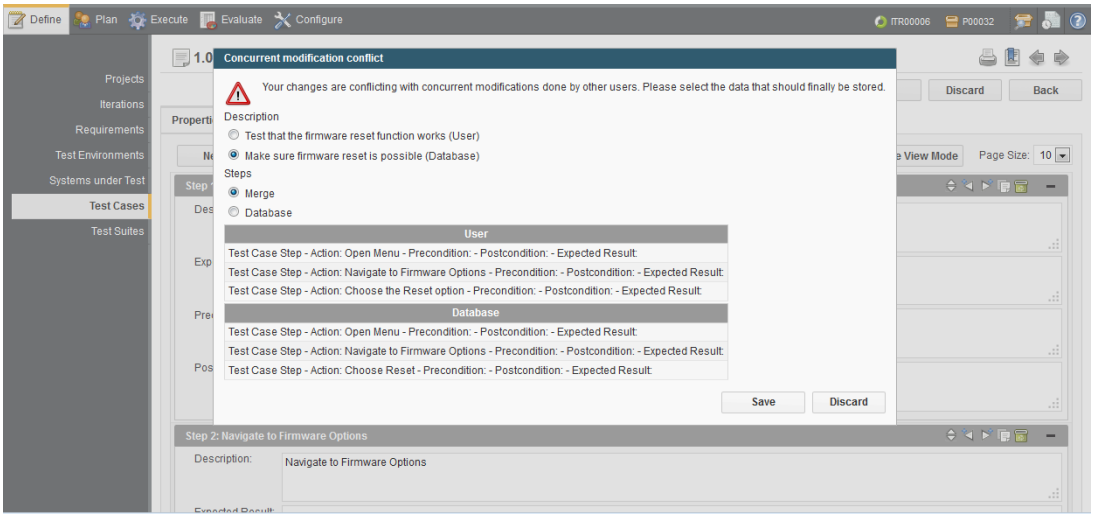

Define
Plan
Execute
Evaluate
Configure
Concurrent modification conflict
Your changes are conflicting with concurrent modifications done by other users. Please select the data that should finally be stored.
Description
Test that the firmware reset function works (User)
Make sure firmware reset is possible (Database)
Steps
Merge
Database
User
Test Case Step - Action: Open Menu - Precondition: - Postcondition: - Expected Result:
Test Case Step - Action: Navigate to Firmware Options - Precondition: - Postcondition: - Expected Result:
Test Case Step - Action: Choose the Reset option - Precondition: - Postcondition: - Expected Result:
Database
Test Case Step - Action: Open Menu - Precondition: - Postcondition: - Expected Result:
Test Case Step - Action: Navigate to Firmware Options - Precondition: - Postcondition: - Expected Result:
Test Case Step - Action: Choose Reset - Precondition: - Postcondition: - Expected Result:
Save
Discard
Back
Step 2: Navigate to Firmware Options
Description:
Navigate to Firmware Options

Figure 5.16. The Conflict Resolution Dialog

For each conflict, the user has the choice of saving the artifact with the value they previously chose or the current value in the database. In the case of lists, e.g. the list of test steps in a test case, the user has the option to *Merge* the changes or to use the version which is currently in the database. This is to prevent the inadvertent loss of data through the deletion of objects which were added by other users.

The *Merge* option builds a list containing all of the elements of both lists (user and database) in an approximation of the correct order.

Clicking the **Save** button persists the chosen changes to the database, and clicking the **Discard** button discards the user's changes.

5.2.9. Deleting Objects

Many objects in Klaros-Testmanagement can be deleted by pressing the icon. If the object to be deleted is linked to other objects (e.g. a requirement which is linked to a test case or contained in other objects (e.g. a test case which is included in a test suite), a popup will be displayed before deleting the object (see Figure 5.17).

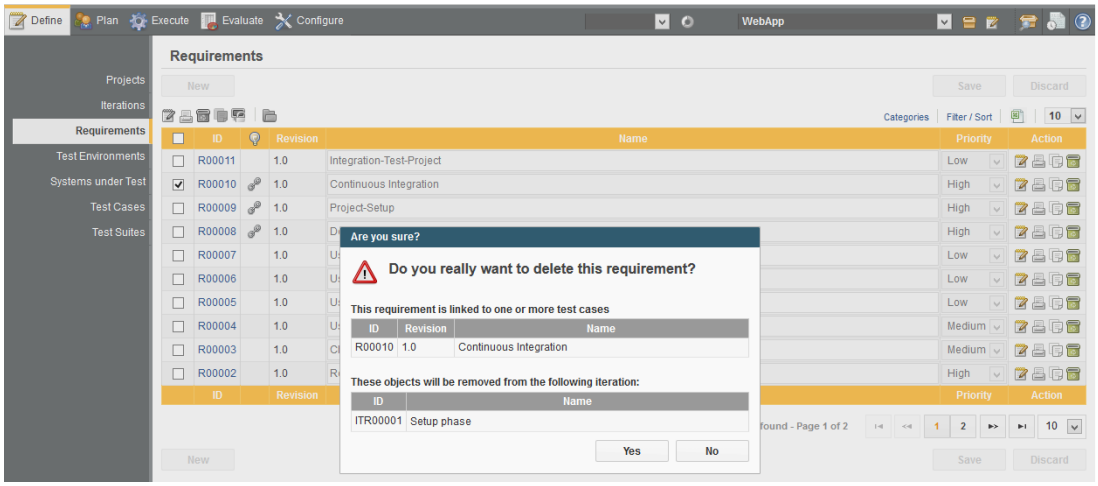


Figure 5.17. The Delete Objects Popup

5.2.10. Bulk Actions

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

Pages in Klaros-Testmanagement which allow editing of a list of objects often also allow bulk actions on these objects. These allow the user to select multiple entries in the list and, e.g., display the print page for them or delete, duplicate or revision them. Assigning objects to a category is also carried out through a bulk action.

In order to use a bulk action, the user must first select one or more objects in the list using the check-boxes to the left of the table. The buttons above and to the left of the table then become active. Figure 5.18 shows the bulk action buttons on the test case list page.

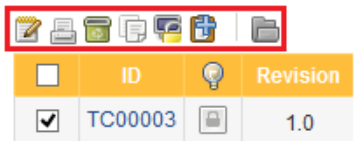


Figure 5.18. The Bulk Actions on the Test Case List Page

Before any bulk action is carried out all changes must be saved. A save changes dialog appears if a bulk action button is pressed while there are unsaved changes.

A bulk action confirmation dialog is displayed once a bulk action button has been clicked and any changes have been saved. These dialogs describe the action which will be carried out and sometimes contain input fields, e.g. to enter the revision comment for the bulk new revision action. All changes are written to the database as soon as this dialog is confirmed.

5.2.10.1. Bulk Edit

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

It is possible to bulk edit objects in Klaros-Testmanagement. Pressing the bulk action icon opens up a popup where fields of multiple objects can be edited at once (See Figure 5.19).

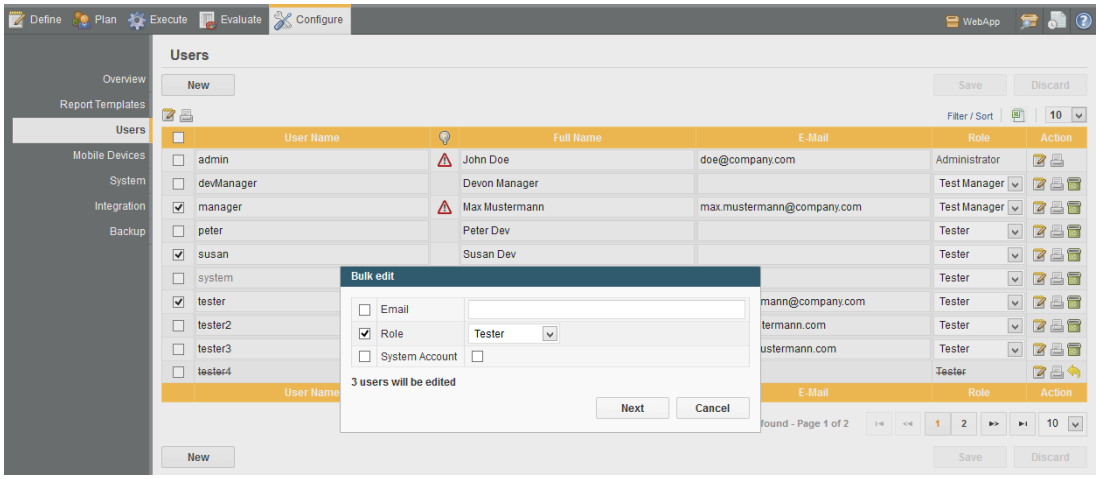


Figure 5.19. Bulk Editing Objects

5.2.11. User Defined Properties

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

In Klaros-Testmanagement, *Iterations*, *Requirements*, *Test Environments*, *Systems under Test*, *Test Cases*, *Test Suites* and *Test Runs* can all be extended by extra fields using user defined properties. These are defined on a per project basis in the *User Defined* tab of the *Project Details* page. This tab is shown in Figure 5.20.

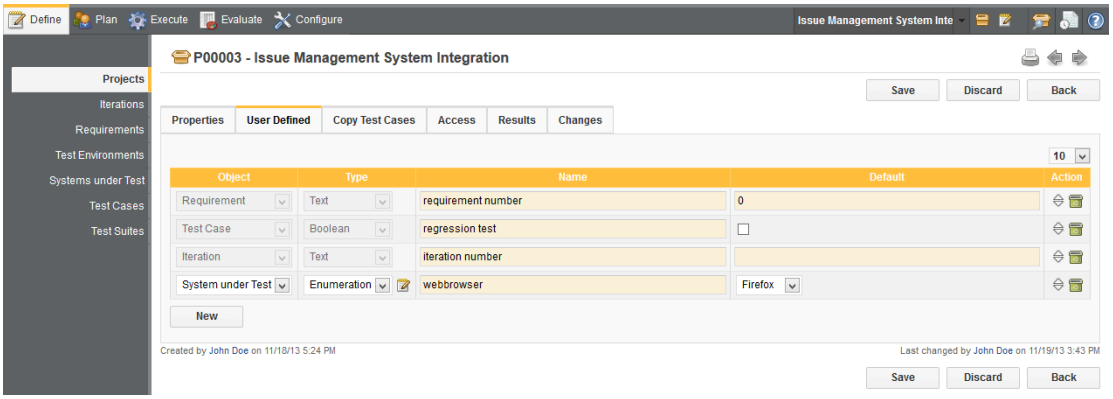


Figure 5.20. The User Defined Tab

Pressing the **New** button adds a new user defined property to the list. After adding a new property, the entity and property type must be chosen.

There are seven possible entity types:

- Iteration
- Job
- Requirement
- System under Test
- Test Environment
- Test Case
- Test Suite
- Test Run

The four possible property types are:

Text	The property will be a single-line text field
Text Field	The property will be a multi-line text field
True/False	The property will be a simple check box
Enumeration	The property will be a drop down box to select a single entry from a predefined list of values

The default value of the property can also be set here.

After the new property is saved the entity type and property type can no longer be changed. The name of the property and the list of enumeration values can always be changed.

5.2.11.1. Enumeration Values

The **Edit** button beside the enumeration property names opens the menu to edit the values of the enumeration property as shown in Figure 5.21

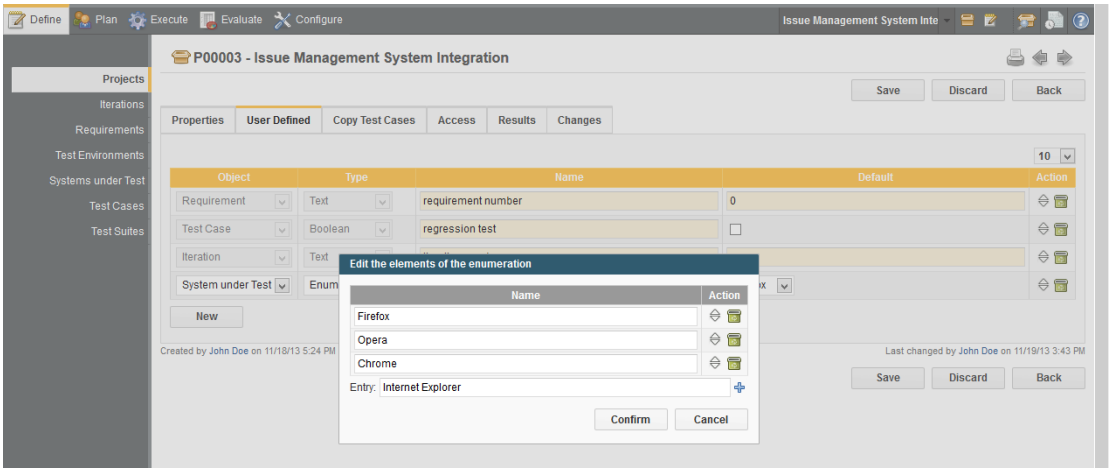


Figure 5.21. Editing an Enumeration Property

Once user defined properties are configured, they can be edited in the *User Defined* tab of the relevant object (Figure 5.22).

Note

If no user defined properties are configured for the object type, this tab will only contain a link to the project's user defined tab in order to create some.

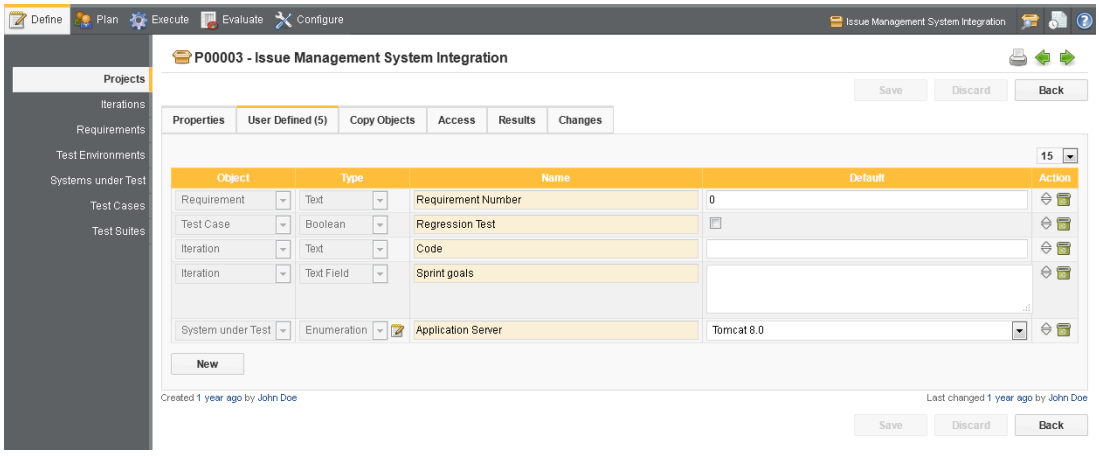


Figure 5.22. The Test Suite User Defined Tab

The values of user defined properties for test runs are set in the execute dialog as shown in Figure 8.3.

5.2.12. Attachments

Attachments may be uploaded and linked to certain objects within Klaros-Testmanagement. This can be done within the *Attachments* tab on the relevant details page or during execution of a test case. Clicking the **Upload Attachment** button opens the upload attachment dialog. Clicking the *Browse...* button opens the *Open File* dialog, which allows the user to select a file to upload. Clicking the **Upload** button uploads the selected file to Klaros-Testmanagement.

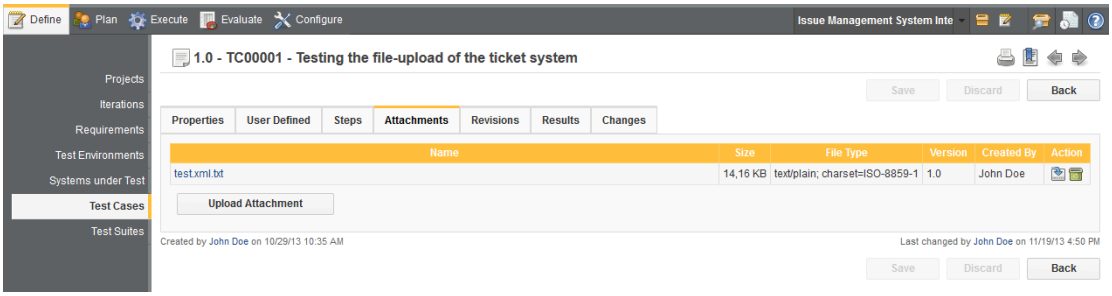


Figure 5.23. The Assign Attachments Screen

After the attachment is saved, it is possible to download the file by clicking on the icon.

Clicking on the icon causes the attachment to be removed from the Klaros-Testmanagement database. The *Save* button must be pressed to persist this change.

5.2.13. Revisions

Requirements, *Test Cases* and *Test Suites* are revisionable artifacts.

A new revision of an object should be created when a major edit is taking place. An old revision of a test case can, for example, be executed when using a system under test where newly added test steps are not applicable.

Revisioning of objects can be managed from the **Revisions** tab. This tab shows the revision history for the object, allows the user to change the revision comment associated with each revision and also allows creation of new revisions. The revision history table may also be used to select a revision to display and edit.

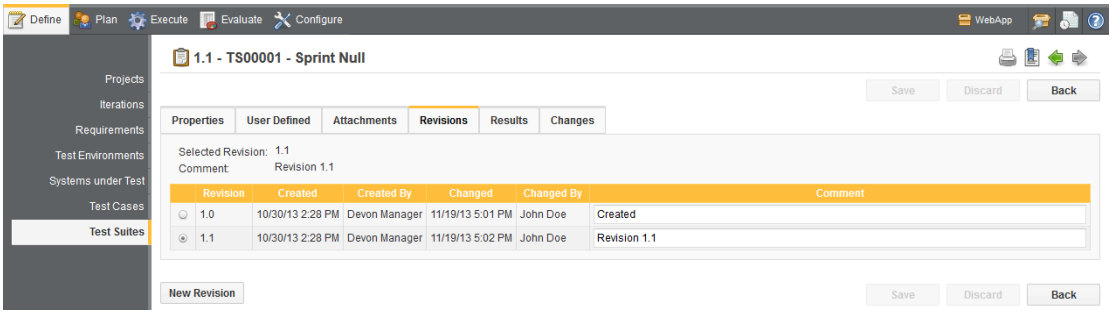


Figure 5.24. The Test Suite Revisions Tab

5.2.14. Related Test Runs and Results

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

Some *Details* sections show test runs and test results which are related to the artifact being displayed, e.g. executed in the displayed test environment or executed by the displayed user. An example of a *Test Results* tab is shown in Figure 5.25.

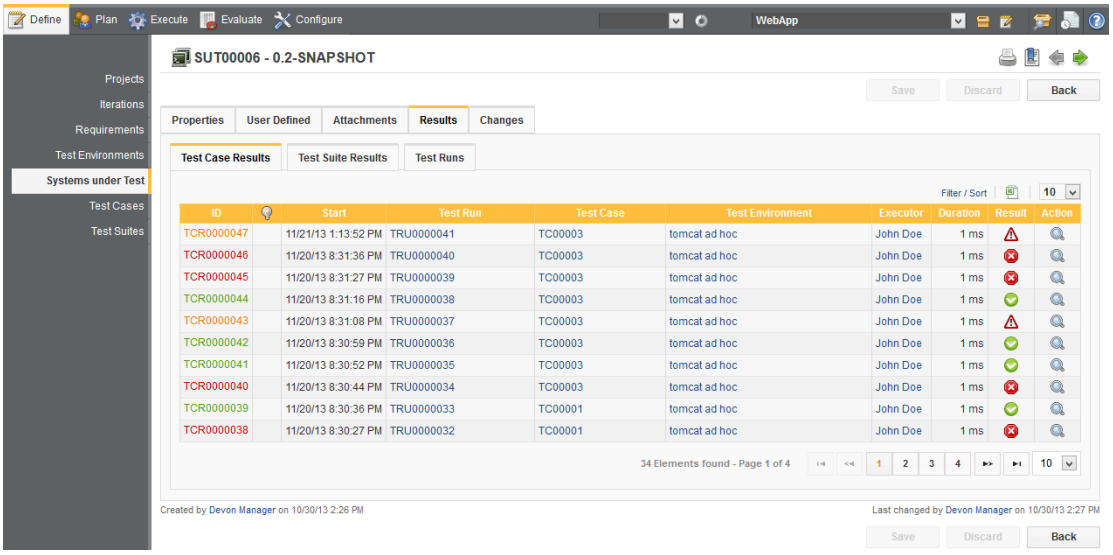


Figure 5.25. The Results Tab

In both of these tabs, the names of artifacts are links to the *Details Pages* of the artifacts. The icon displays the *Details Page* of the corresponding test run or test result.

In the *Test Runs* tab (Figure 5.25), reports may also be generated, as described in Section 9.4, "Test Runs".

5.2.15. Change History

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

The *Details* section of each editable object in Klaros-Testmanagement contains a *Changes* tab. This shows the change history for the object.

Figure 5.26 shows the *Changes* tab for a test case.

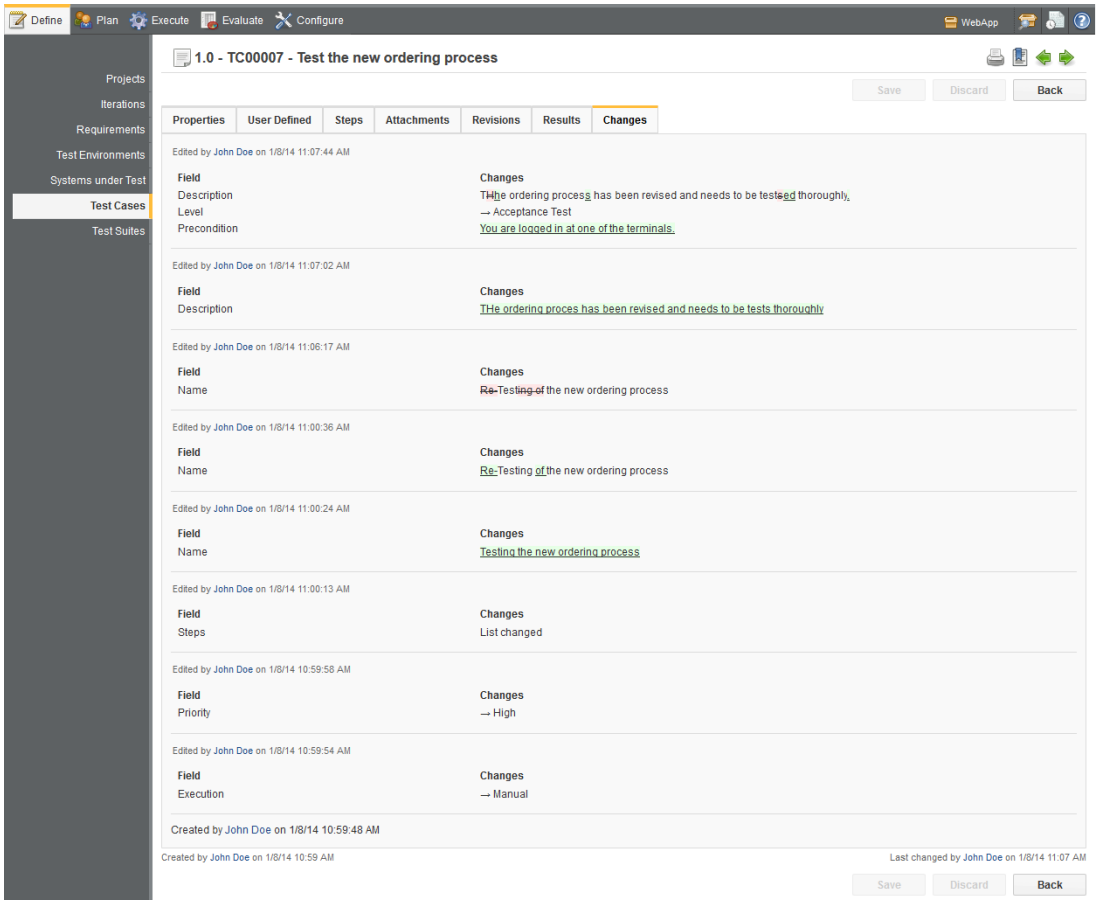


Figure 5.26. The Test Case Changes Tab

Changes to an object are grouped by edit, i.e. if a user changes two fields and then presses save, the changes to these to fields will be grouped in the *Changes* tab.

5.2.16. Overview Tabs

Feature only available in Klaros-Testmanagement Enterprise Edition

For every type of artifact in Klaros-Testmanagement there is a dedicated overview tab which shows the most important data for this artifact (like the last editor or the the status) at a glance. In addition to raw data, many overview pages are enriched with various useful statistics, like for example the success history for test suites, which helps visualizing the test progress.

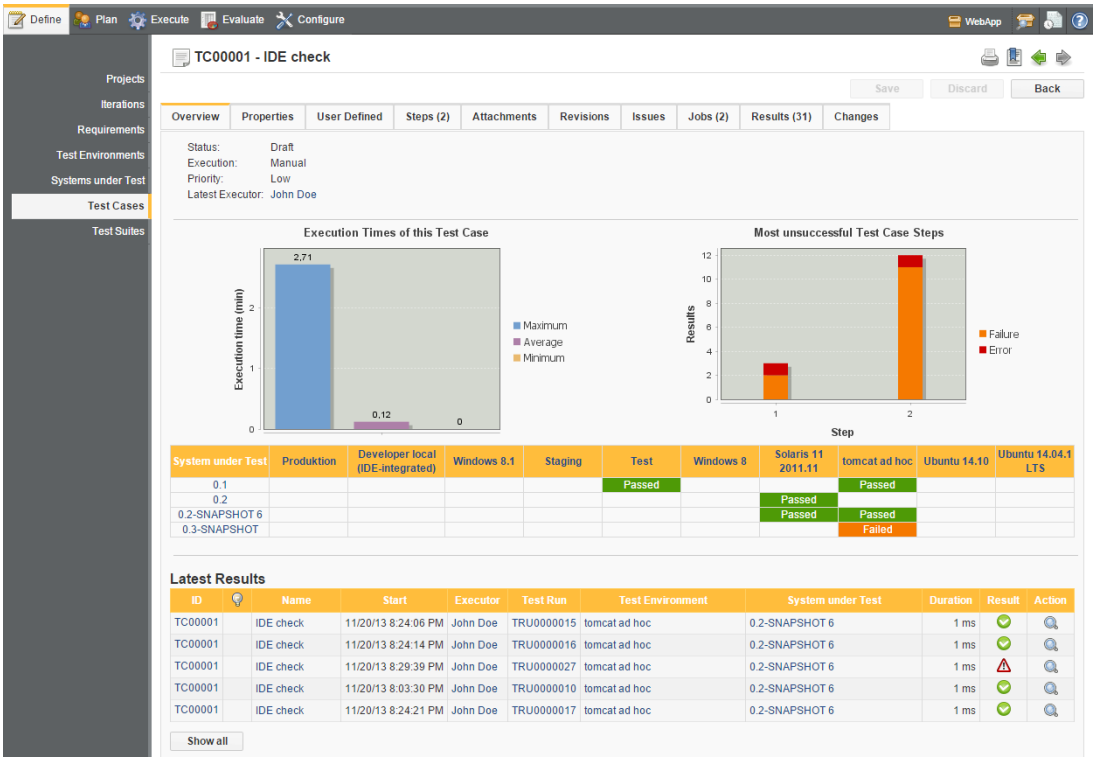


Figure 5.27. The Test Case Overview Tab

Chapter 6. Define

This section describes the *Define* section of Klaros-Testmanagement. In this section, Projects, Iterations, Requirements, Test Environments, Systems under Test, Test Cases and Test Suites may be created and edited. There is a menu entry on the left-hand side for each of these artifact types.

6.1. Projects

The *Projects* section allows managing projects in Klaros-Testmanagement. Projects are collections of all test artifacts for a given product and the related results and jobs. Test artifacts are objects like test cases and test suites.

When accessing Klaros-Testmanagement for the first time it is necessary to create a project to work with. To create a new project select the menu entry *Projects* (Figure 6.1) and follow Section 6.1.1, “Creating a Project”.

After at least one project has been created, users may select one from the table.

After a successful login the *Projects* screen is shown. It gives the user an overview of the projects stored in the Klaros-Testmanagement database. Projects may be created, removed and updated on this page (Figure 6.1). The displayed projects may be filtered and sorted as described in Section 5.2.5, “Filtering and Sorting”.

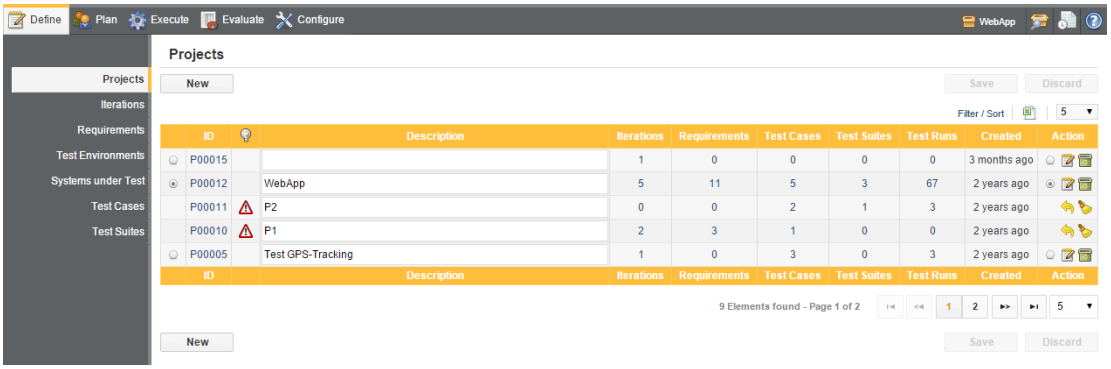


Figure 6.1. The Projects Screen

In order to use the test management functions of Klaros-Testmanagement, the user is required to select a project. Projects are selected by clicking on the check box of the desired project in the leftmost or rightmost column.

Choosing a project with the project quick-select drop-down field

The header bar of Klaros-Testmanagement contains a drop-down field labeled *Project*. With this drop-down field the active project can be switched quickly on any Klaros-Testmanagement page. You will be directed to the top page of your current category (Define / Execute etc.).

If no projects have been created, the user must create one before proceeding. The process of creating a project is described in the next section.

After a project is selected, the disabled functions in the navigation bar are made available.

6.1.1. Creating a Project

New projects may be created by clicking the **New** button (Figure 6.1). An empty row is added at the beginning of the table. The *Project ID* is automatically assigned by Klaros-Testmanagement, and the description can be specified.

6.1.2. Deleting a Project

When the Delete icon is clicked, the project will be locked and removed from the list of selectable projects.

Why is the Delete button sometimes disabled?

Projects that already contain some test results can no longer be deleted by users, so their delete icon is disabled ( ). Only an administrator can delete a locked project. Locked projects may be restored as long as they have not yet been purged from the database.

6.1.3. Restoring a Project

When the restore icon is clicked, a previously locked project is restored to the list of selectable projects. Only an administrator is able to restore a locked project.

6.1.4. Purging a Project

When the purge icon is clicked, a previously locked project is purged from the database. Only an administrator is able to purge a locked project. This operation cannot be undone.

6.1.5. Editing the Description of a Project

The project description can be used to give a short overview over the project. Clicking on the *Description* field allows the user to directly change the current value (Figure 6.1).

6.1.6. Project Details

This section allows in-depth editing of an individual project, including adding/removing issue management systems and user-defined properties. The executed test results and the change history of this project are also visible in this section.

6.1.6.1. Editing the Issue Management Systems in a Project

The list of issue management systems that are in use in the project can be edited on the project details page, which may be reached by pressing the ( ) edit icon beside a project on the projects page (Figure 6.1).

Creating issue management systems

Before issue management systems are available for assignment, they must be created by an administrator.

For instructions on the creation of issue management systems see Section 10.6.1, “Issue Management”.

The project properties page in the project details section (Figure 6.2, “The Project Properties Page”) shows two tables: The upper table contains the issue management systems that are already in use in the project, the lower table shows all issue management systems that are configured in Klaros-Testmanagement and not in use by the project. Pressing the add icon adds the corresponding issue management system to the project. The issue management system is removed from the lower table and added to the upper table.

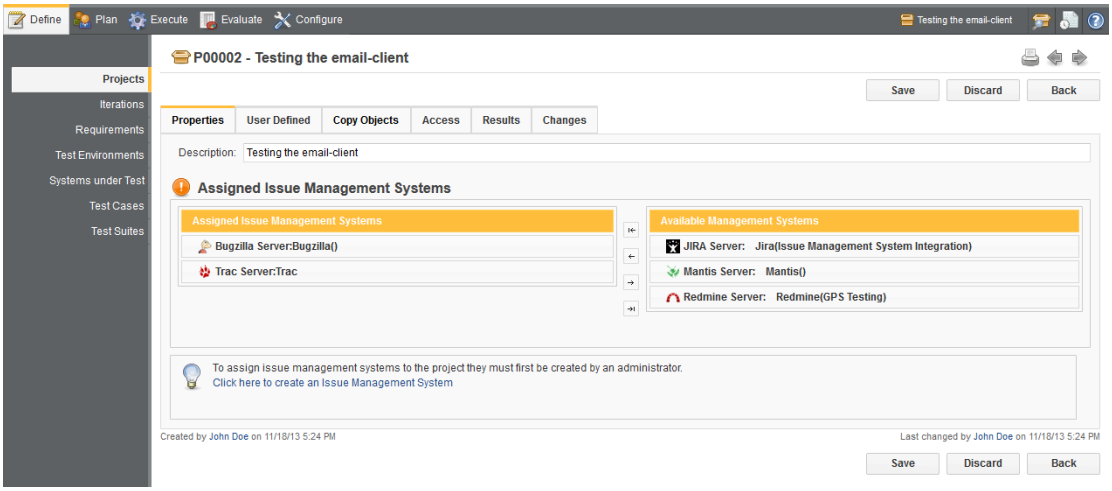


Figure 6.2. The Project Properties Page

On this page it is also possible to edit the *Description* of the project.

6.1.6.2. User Defined Properties

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

User defined properties can be used to customize objects within Klaros-Testmanagement by adding custom fields. This process is described in Section 5.2.11, “User Defined Properties”.

6.1.6.3. Copying Objects from another Project to the current Project

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

It is possible to copy objects from another project to the current one. This function can be accessed via the *Copy Objects* tab on the project details page (see Figure 6.2, “The Project Properties Page”). To copy objects, a project has to be selected from the *Source Project* drop down box first. This opens up several tabs with requirements, test environments, systems under test, test cases and test suites from the source project.

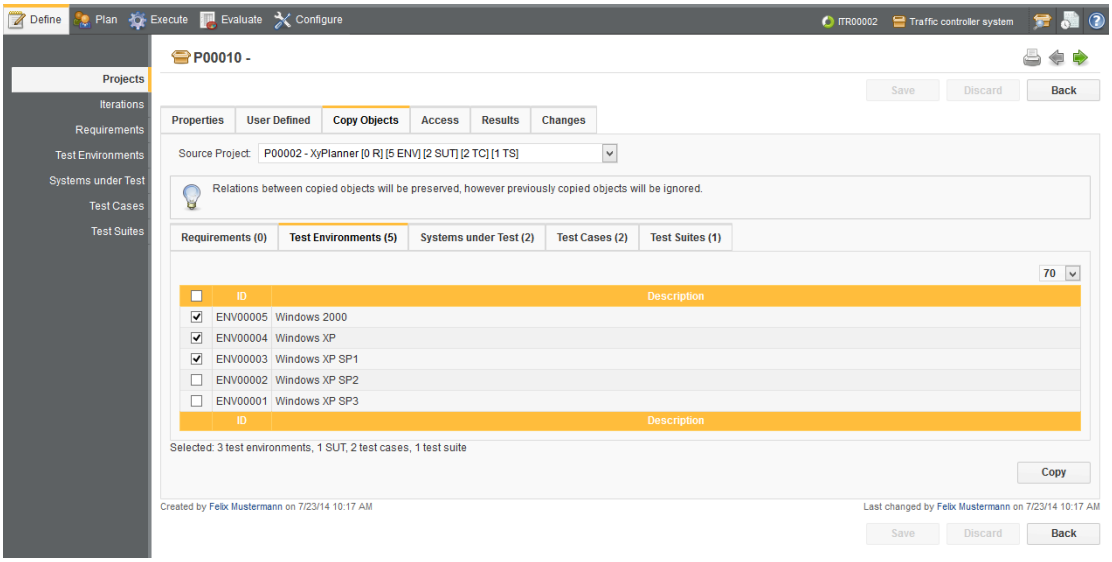


Figure 6.3. Copying Objects to the Current Project

After selecting the desired objects, pressing the **Copy** button opens up a popup with a summary of all objects that are going to be copied to the current project. Pressing the *yes* button closes the popup and starts the copy operation (see Figure 6.4, “Copying Objects from another Project”).

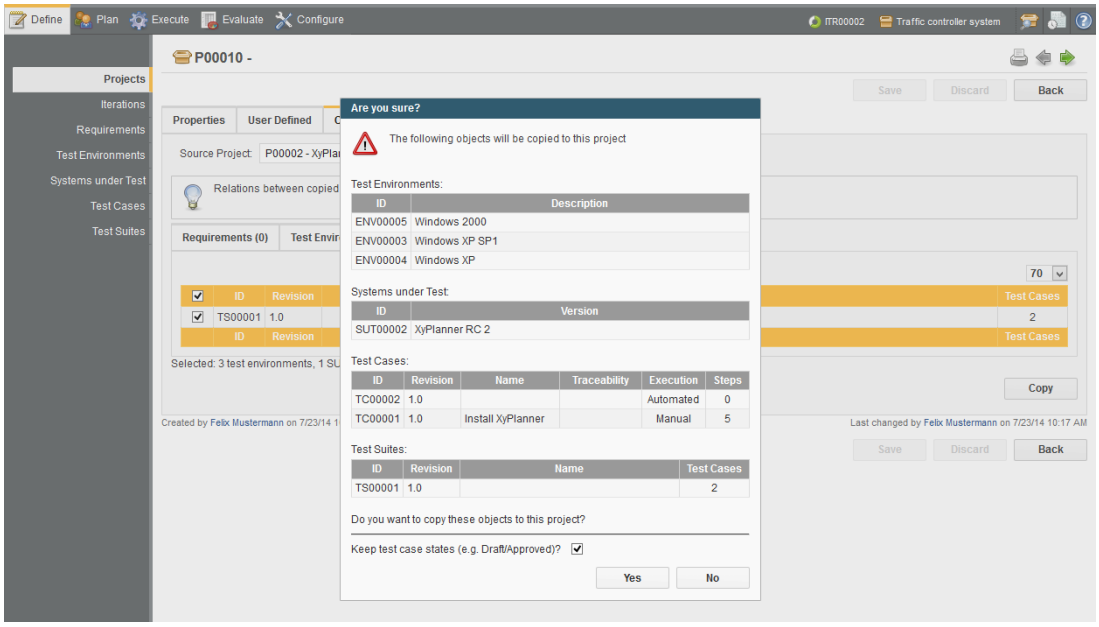


Figure 6.4. Copying Objects from another Project

Checking the *Keep test case states* checkbox keeps the state of all test cases while copying. Unchecking this checkbox sets all test case states to *Draft*.

Note

When copying a test suite, all test cases that are part of this test suite are also being copied.

6.1.6.4. Securing access to the Project

A project can be secured by checking the *Disable anonymous project access* flag. If this flag is checked, a user must be logged into Klaros-Testmanagement to view bookmarked pages. More information about bookmarking is available at Section 9.8, “Bookmarking Results”.

6.1.6.4.1. Assigning Project specific Roles

Feature only available in Klaros-Testmanagement Enterprise Edition

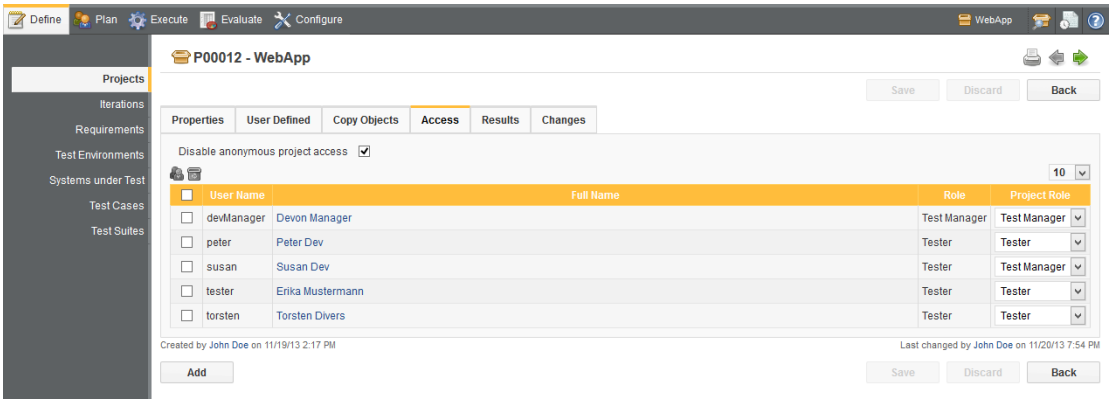


Figure 6.5. The Project Access tab

Klaros-Testmanagement Enterprise Edition integrates a rights and role system with its user management system (see Section 10.3, “Users”). In the *Access* tab administrators and test managers can assign and change project roles for individual users. For example, a user which has the global role tester can be assigned to one project as a tester and to another project as a test manager.

Initially, every project can be accessed by every user. If at least one tester or test manager has been assigned to a specific project, access to this project is restricted for all other users without an assigned role.

Pressing the **Assign** button opens up a popup (see Figure 6.6, “The Project Access Pop-up”) where new users can be added to the project.

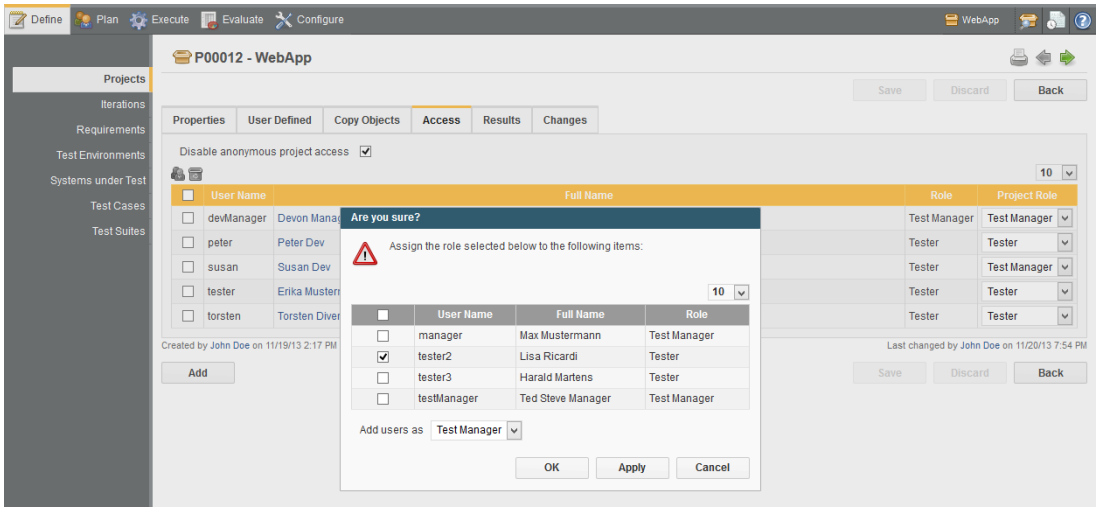


Figure 6.6. The Project Access Popup

Every project needs at least one test manager

A project needs at least one assigned test manager. When trying to save a project which has no assigned test manager, a warning message will be displayed in the log panel (see Section 5.2.4, “The Log Panel”).

6.1.6.5. Results

The result tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab showing the test results related to this project as described in Section 5.2.14, “Related Test Runs and Results”.

6.2. Iterations

Feature only available in Klaros-Testmanagement Enterprise Edition

Iterations represent a test cycle in a project. This helps synchronizing the test process with an agile development process (e.g. Scrum). The *Iterations* page allows management of the iterations in the selected project.

An iteration can be selected for the current project by selecting it in the toolbar on the top of the page (see Figure 6.7, “The Iteration and Project Selection Toolbar”). When an iteration is active, only test objects relevant to the test cycle represented by the iteration are visible. All test runs executed while an iteration is active are automatically linked to the iteration. This helps to quickly assess the test progress and success rates for a given test cycle.

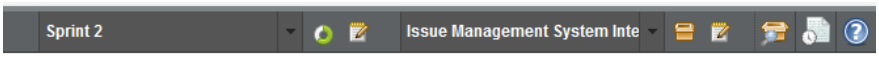


Figure 6.7. The Iteration and Project Selection Toolbar

Iterations may be created, removed, updated and assigned to categories on this page (Figure 6.8). The displayed iterations may be filtered and sorted as described in Section 5.2.5, “Filtering and Sorting”.

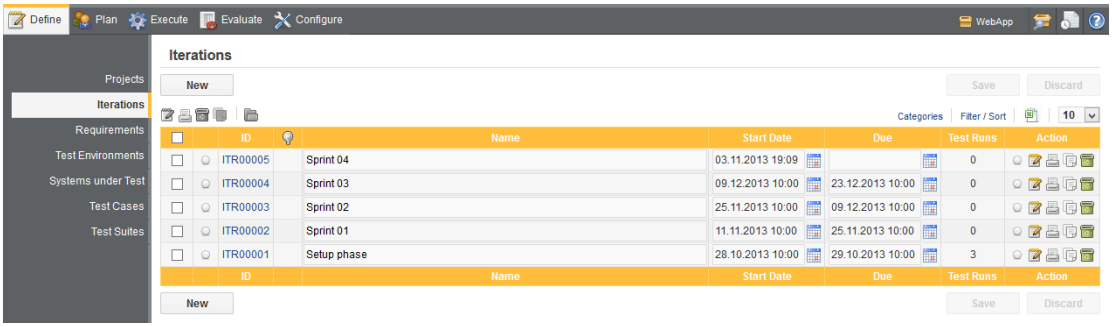


Figure 6.8. The Iterations Screen

The *Iterations* section shows all of the iterations defined for the current project. Clicking on the table fields allows direct editing of the attributes.

The leftmost column allows iterations to be selected for bulk actions. The uppermost check-box is a global select button, which selects or deselects all elements shown in the table. Please note that the displayed elements are subject to the current search and catagorization criteria.

Iterations may be viewed on a printer-friendly page as described in Section 5.2.6, “Print Pages” as well as categorized as described in Section 5.2.7, “Categorization”.

Clicking on the delete icon removes the Iteration.

Why is the Delete button sometimes disabled?

Iterations that are referenced by test runs cannot be deleted, so their delete icon is disabled ( ).

Clicking on an edit icon displays the *Iteration Details* page of the corresponding iteration.

6.2.1. Bulk Actions

Some bulk actions are available on the *Maintain Iterations* page. Bulk actions are described in Section 5.2.10, “Bulk Actions”.

The bulk actions available on this page are: *Edit*, *Print*, *Delete* and *Assign to Category*. The bulk action buttons can be found above the table on the left.

6.2.2. Creating an Iteration

Clicking on the **New** button creates a new iteration in the currently selected project (Figure 6.8). The *Name*, *Start Date* and *Due Date* fields can then be specified. The iteration ID is automatically assigned by Klaros-Testmanagement.

6.2.3. Iteration Details

6.2.3.1. Iteration Overview

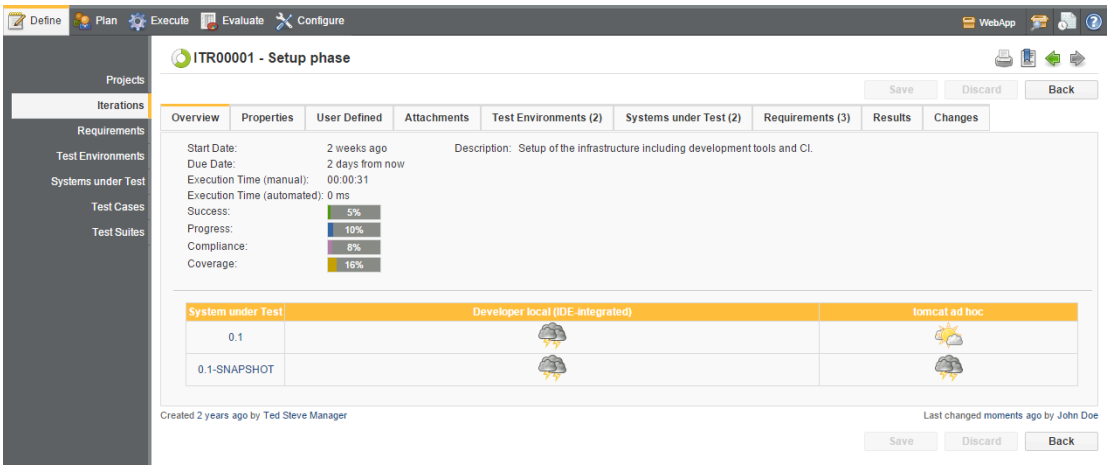


Figure 6.9. The Iteration Overview Tab

- *Start Date*

  The point in time when the iteration starts.

- *Due Date*

  The point in time when the iteration ends.

- *Execution Time (manual)* and *Execution Time (automated)*

  The overall execution times from manual and automated test executions of the iteration are summed up and shown as "execution time (manual)" and "execution time (automated)".

Tip

Those values can be used for further investigation. An extremely high manual execution time for example could indicate a high rate of retesting or a need for more automated test cases.

- *Success*

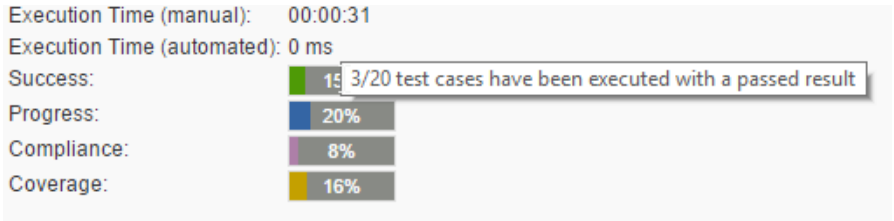


Figure 6.10. The Iteration Success Rate

All test cases which are included in this iteration contribute to this metric (if none are explicitly registered for this iteration, then all test cases are considered). A success rate of 100% means that the latest test run for every test case has been successful.

Note

Even if a test case has been successfully executed in the past, only the latest result is considered for the success rate.

- *Progress*

  The progress rate shows how many test cases which are considered for this iteration have been executed.

Note

In contrast to the success rate, the progress rate considers only if a test case has been executed at least once in the iteration, regardless of the result.

- *Compliance*

  The compliance rate shows how many test cases, which are part of this iterations requirements, have been successfully executed.

- *Coverage*

  The coverage rate shows how many test cases, which are part of this iteration's requirements, have been executed at least once.

Note

In contrast to the compliance rate, the coverage rate considers only if a test case has been executed at least once in the iteration, regardless of the result (passed, failed, error).

- *Health Matrix*

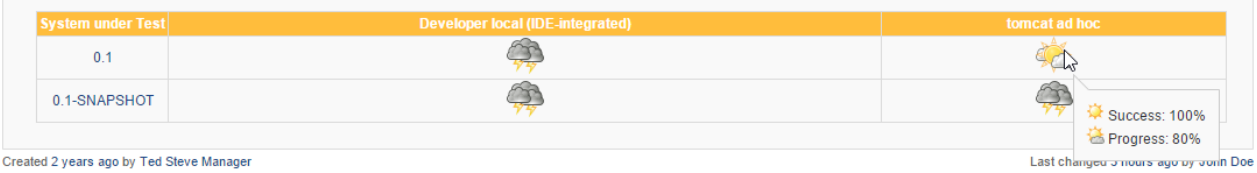


Figure 6.11. The Iteration Health Matrix

This matrix shows the success and progress rate of this iteration for every combination of test environment and system under test which have been configured for this iteration. Both rates are combined to form a weather icon which shows a sun if both the success and progress rate are at 100 percent and a thundery cloud if both rates are below a ceratin

level. Moving the mouse cursor over the weather icon displays the success and progress rate in detail.

Note

There are various gradations inbetween the sun and the thundery cloud which can be used to show the condition of the system under test and test environment in this iteration in finer granularity. This can be configured via the health matrix on the reports dashboard.

6.2.3.2. Iteration Properties

This tab (Figure 6.12) allows the user to view or change the following attributes of the selected iteration:

- *ID*

  Assigned automatically.

- *Name*

  The editable name of the iteration.

- *Description*

  A description of the iteration.

- *Success Criteria*

  A description of the conditions which need to be met for this iteration to be complete.

- *Start Date*

  The date on which this iteration starts.

- *Due Date*

  The date on which this iteration is due to be finished.

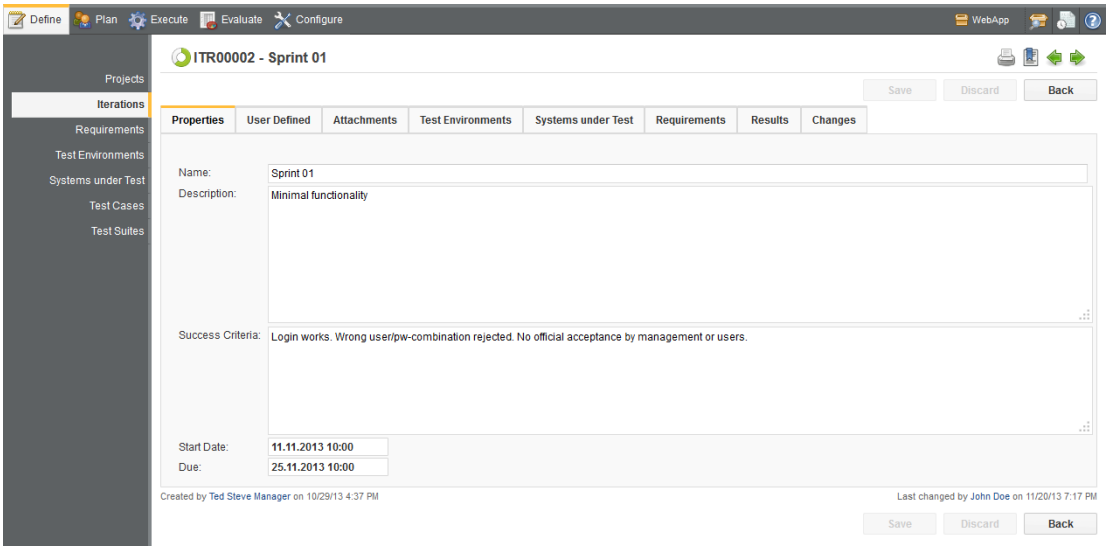


Figure 6.12. The Iteration Details Screen

6.2.3.2.1. User Defined Properties

User defined properties can be used to customize iterations within Klaros-Testmanagement by adding custom fields. This process is described in Section 5.2.11, “User Defined Properties”.

6.2.3.2.2. Attachments

Attachments may be added to iterations. This process is described in Section 5.2.12, “Attachments”.

6.2.3.2.3. Test Environments, Systems under Test and Requirements Tabs

These tabs allow linking of other artifacts to the iteration. Simply click the **Assign** button to open a dialog which allows you to select artifacts to assign to this iteration.

Artifacts may be removed by selecting them and clicking the icon above the table.

The *Test Environments* tab shows the test environments which have been added to this iteration.

The *Systems under Test* tab shows the systems under test which have been added to this iteration as well as their *Compliance*, *Coverage*, *Success* and *Progress* metrics. These are metrics relating to the amount of test cases executed and the latest results of these executions.

Progress and *Success* show these metrics for all test cases in the project, while *Coverage* and *Compliance* show them for test cases which cover requirements in this iteration.

Progress — The percentage of the test cases in the project which have been executed in this iteration.

Success — For the test cases executed in the *Progress* metric, the percentage which were last executed with the result *Passed*.

Coverage — The percentage of the test cases which cover requirements which are included in this iteration which have been executed in this iteration.

Compliance — For the test cases executed in the *Coverage* metric, the percentage which were last executed with the result *Passed*.

The *Requirements* tab shows the requirements which have been added to this iteration as well as their *Compliance* and *Coverage* metrics. These are metrics relating to the amount of test cases executed against the requirements and the latest results of these executions.

Coverage — The percentage of the test cases which cover the requirement which have been executed in this iteration.

Compliance — For the test cases executed in the *Coverage* metric, the percentage which were last executed with the result *Passed*.

6.2.3.2.4. Results

The result tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab showing the test results related to this iteration as described in Section 5.2.14, “Related Test Runs and Results”.

6.2.3.2.5. Bookmarking Iterations

Iterations may be linked to from outside of Klaros-Testmanagement using the link on the bookmark icon. Right-click the icon and choose *Copy Link Location* (Firefox/Chrome) or the equivalent option.

6.3. Requirements

Feature only available in Klaros-Testmanagement Enterprise Edition

Requirements are conditions which must be met in order for the product to be considered ready to be shipped. The *Requirements* section allows management of the requirements in the selected project.

Requirements may be linked to test cases. This enables users to quickly check the status of a given requirement.

Requirements may be created, removed, updated and assigned to categories on this page (Figure 6.13). The displayed requirements may be filtered and sorted as described in Section 5.2.5, “Filtering and Sorting”.

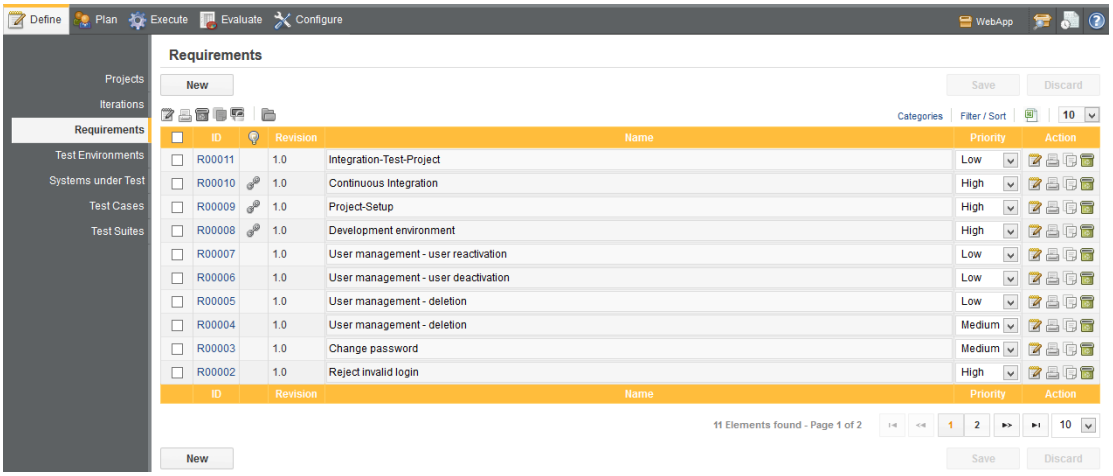


Figure 6.13. The Requirements Screen

The *Requirements* page shows all of the requirements defined for the current project. Clicking on the table fields allows direct editing of the attributes.

The leftmost column allows requirements to be selected for bulk actions. The uppermost checkbox is a global select button, which selects or deselects all elements shown in the table. Please note that the displayed elements are subject to the current search and catagorization criteria.

Requirements may be viewed on a printer-friendly page as described in Section 5.2.6, “Print Pages” as well as categorized as described in Section 5.2.7, “Categorization”.

Clicking on the delete icon removes the requirement.

Why is the Delete button sometimes disabled?

Requirements that are referenced by test runs cannot be deleted, so their delete icon is disabled ( ).

Clicking on an edit icon displays the *Requirement Details* page of the corresponding requirement.

6.3.1. Bulk Actions

Some bulk actions are available on the *Requirements* page. Bulk actions are described in Section 5.2.10, “Bulk Actions”.

The bulk actions available on this page are: *Edit*, *Print*, *Delete*, *New Revision*, *Duplicate* and *Assign to category*. The bulk action buttons can be found above the table on the left.

6.3.2. Creating a Requirement

Clicking on the **New** button creates a new Requirement in the currently selected project (Figure 6.13). The *Name* and *Priority* fields can then be specified. The requirement ID is automatically assigned by Klaros-Testmanagement.

6.3.3. Requirement Details

6.3.3.1. Requirement Overview

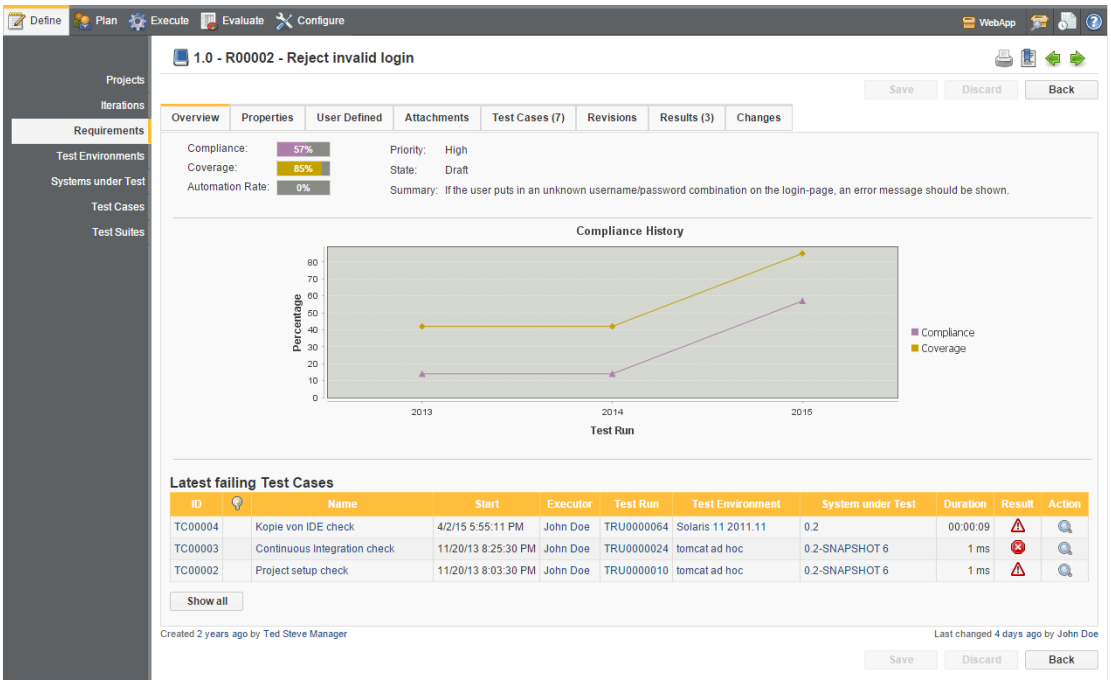


Figure 6.14. The Requirement Overview Tab

- *Priority*

  The priority of the requirement. This signifies how crutial the compliance of this requirement is.

- *Compliance*

  The compliance rate shows how many test cases, which are part of this requirement, have been successfully executed.

- *Coverage*

  The compliance rate shows how many test cases, which are part of this requirement, have been executed at least once.

Note

Note: In contrast to the success rate, the coverage rate considers only if a test case has been executed at least once in the iteration, regardless of the result (passed, failed, error).

- *Automation Rate*

  This bar shows how many of the test cases of this requirement are automated.

Tip

Manual test cases can often be converted to automated test cases if they are frequently executed to save time.

- *Compliance History*

  This graph shows how the compliance and the coverage for this requirement have changed since the requirement has been defined. The timeline is automatically adjusted depending on the given timeframe.

- *Latest Failing Test Cases*

  This table lists the test cases which recently have been failed during a test run for this requirement alongside with the appertaining test case result.

Note

This table is limited to five entries. The other test case results can be viewed by pressing the *Show All* button.

6.3.3.2. Requirement Properties

This page (Figure 6.15) allows the user to view or change the following attributes of the selected requirement:

- *ID*

  Assigned automatically.

- *Name*

  The editable name of the requirement.

- *Priority*

  The priority of the requirement. Can be *Low*, *Medium*, *High* or empty.

- *State*

  The requirement state determines whether it is editable or visible in reports. Only requirement with the state *Draft* are editable. A requirement with the state *Skip* is not shown in any reports.

- *Summary*

  A short summary of the requirement.

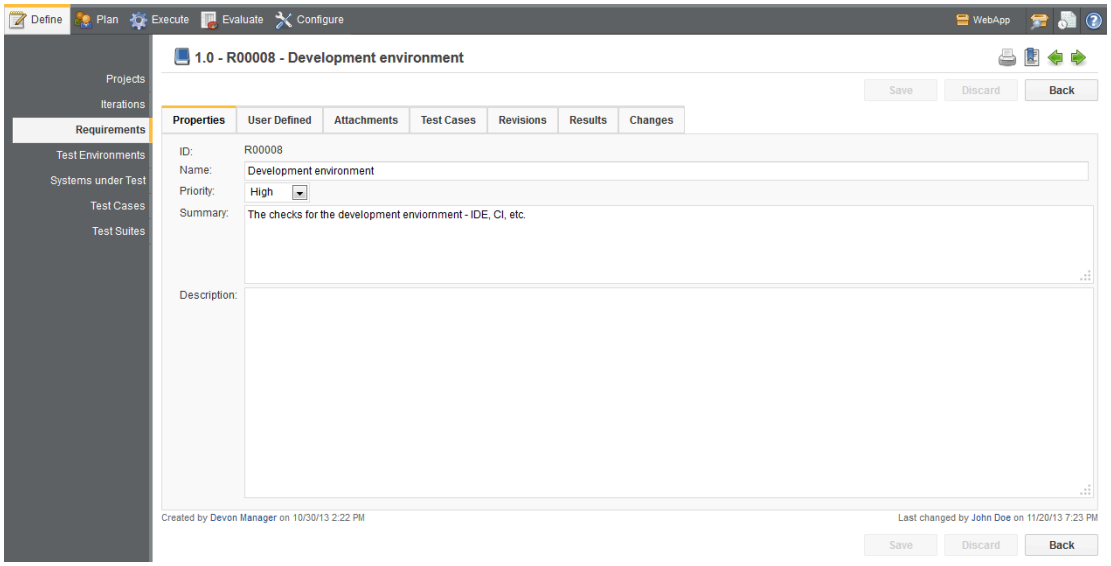


Figure 6.15. The Requirement Details Screen

6.3.3.3. User Defined Properties

User defined properties can be used to customize requirements within Klaros-Testmanagement by adding custom fields. This process is described in Section 5.2.11, “User Defined Properties”.

6.3.3.4. Attachments

Attachments may be added to requirements. This process is described in Section 5.2.12, “Attachments”.

6.3.3.5. Test Cases

Coverage of requirements is tested by test cases. Users can link test cases to requirements in the *Test Cases* tab. The test results of linked test cases count towards the execution and success rates of the requirement.

Test cases may be assigned to the requirement by clicking the **Assign** button. This opens a dialog which allows Test Cases to be selected. Test cases may be removed from the requirement one at a time by clicking the icon in the *Action* column or in bulk by selecting test cases and clicking the icon above the table.

6.3.3.6. Revisions

Requirements are revisionable. The process of revisioning an object is described in Section 5.2.13, “Revisions”.

6.3.3.7. Bookmarking Requirements

Requirements may be linked to from outside of Klaros-Testmanagement using the link on the bookmark icon. Right-click the icon and choose *Copy Link Location* (Firefox/Chrome) or the equivalent option.

6.4. Test Environments

The *Test Environments* section allows the management of the test environments in the selected project. Test environments represent the extrinsic settings that may influence the test result. Examples for components of a test environments are the operating system or an application server (e.g. Tomcat 7 on Ubuntu 12.10).

Test environments may be created, removed, updated and assigned to categories on this page (Figure 6.16). The displayed test environments may be filtered and sorted as described in Section 5.2.5, “Filtering and Sorting”.

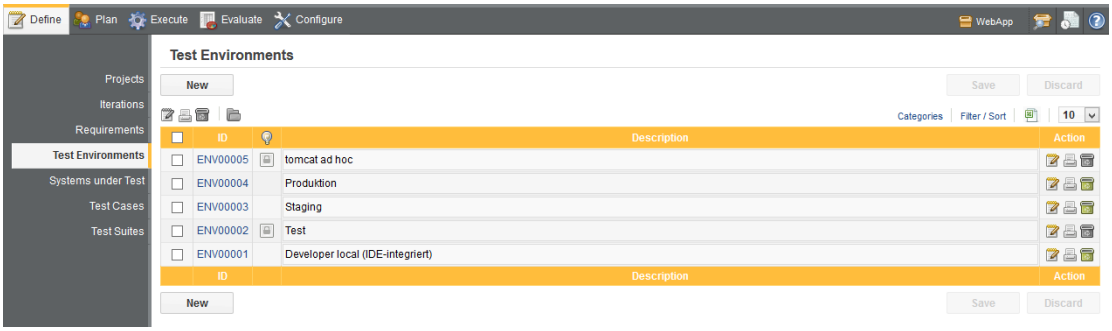


Figure 6.16. The Test Environments Screen

The *Test Environments* page shows all of the test environments defined for the current project. Clicking on the table fields allows direct editing of the attributes.

The leftmost column allows test environments to be selected for bulk actions. The uppermost checkbox is a global select button, which selects or deselects all elements shown in the table. Please note that the displayed elements are subject to the current search and catagorization criteria.

Test Environments may be viewed on a printer-friendly page as described in Section 5.2.6, “Print Pages” as well as categorized as described in Section 5.2.7, “Categorization”.

Clicking on the delete icon removes the test environment.

Why is the Delete button sometimes disabled?

Test environments that are referenced by test runs cannot be deleted, so their delete icon is disabled ( ).

Clicking on an edit icon displays the *Test Environment Details* page of the corresponding test environment.

6.4.1. Bulk Actions

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

Some bulk actions are available on the *Environments* page. Bulk actions are described in Section 5.2.10, “Bulk Actions”.

The bulk actions available on this page are: *Edit*, *Print*, *Delete* and *Assign to category*. The bulk action buttons can be found above the table on the left.

6.4.2. Creating a Test Environment

Clicking on the **New** button creates a new test environment in the currently selected project (Figure 6.16). The description field can then be specified. The test environment ID is automatically assigned by Klaros-Testmanagement.

6.4.3. Test Environment Details

This section allows in-depth editing of an individual test environment, including user defined properties. The executed test results and change history for the test environment are also visible in this section.

6.4.3.1. Test Environment Overview

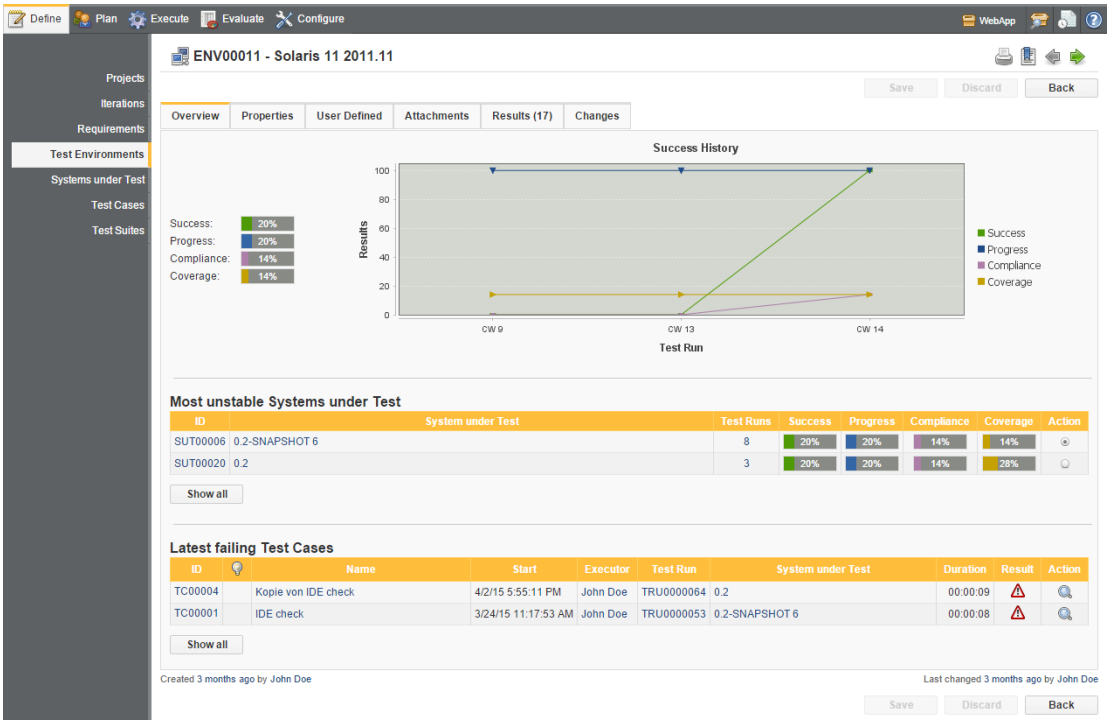


Figure 6.17. The Test Environment Overview Tab

- *Success*

  The success rate shows how many test cases have been successfully executed in this test environment.

- *Progress*

  The progress rate shows how many test cases have been executed at least once in this test environment.

Note

In contrast to the success rate, the coverage rate considers only if a test case has been executed at least once in this test environment, regardless of the result (passed, failed, error).

- *Compliance*

  The compliance rate shows how many test cases, which are part of this iterations requirements, have been successfully executed in this test environment.

- *Coverage*

  The coverage rate shows how many test cases, which are part of a requirement, have been executed at least once in this test environment.

Note

In contrast to the compliance rate, the coverage rate considers only if a test case has been executed at least once in the iteration, regardless of the result (passed, failed, error).

- *Success History*

  This graph shows changes in the compliance, coverage, progress and success rates of this test environment over time. The timeline is automatically adjusted depending on the given timeframe.

- *Most unstable Systems under Test*

  This table displays the systems under test which have the lowest success and progress rates in conjunction with this test environment.

Note

This table is limited to five entries. The other system under tests can be viewed by pressing the *Show All* button.

- *Latest failing Test Cases*

  This table lists the test cases which recently have been failed during a test run in this test environment alongside with the appertaining test case result.

Note

This table is limited to five entries. The other system under tests can be viewed by pressing the *Show All* button.

6.4.3.2. User Defined Properties

Feature only available in Klaros-Testmanagement Enterprise Edition

User defined properties can be used to customize test environments within Klaros-Testmanagement by adding custom fields. This process is described in Section 5.2.11, “User Defined Properties”.

6.4.3.3. Attachments

Attachments may be added to test environments. This process is described in Section 5.2.12, "Attachments".

6.4.3.4. Results

The result tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab showing the test results related to this test environment as described in Section 5.2.14, "Related Test Runs and Results".

6.4.3.5. Bookmarking Test Environments

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

Test environments may be linked to from outside of Klaros-Testmanagement using the link on the bookmark icon. Right-click the con and choose *Copy Link Location* (Firefox/Chrome) or the equivalent option.

6.5. Systems under Test

The systems under test section allows maintenance of the systems under test in the selected project. A system under test is used to represent a version of a software product that can be tested.

The *Systems under Test* page allows the creation, removal and updating of systems under test and assigning them to categories. The displayed systems under test may be filtered and sorted as described in Section 5.2.5, "Filtering and Sorting" (Figure 6.18).

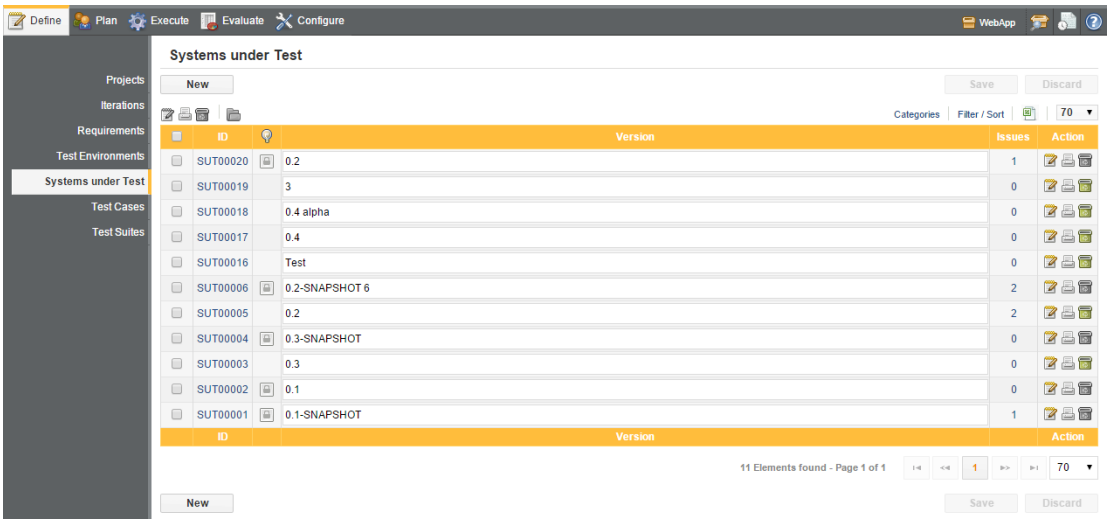


Figure 6.18. The Systems under Test Screen

The *Systems under Test* screen shows a table with all of the systems under test in current project. The version of the system under test can be changed by clicking on the text field and editing it directly.

The leftmost column allows system under test to be selected for bulk actions. The uppermost checkbox is a global select button, which selects or deselects all elements shown in the table. Please note that the displayed elements are subject to the current search and catagorization criteria.

Systems under Test may be viewed on a printer-friendly page as described in Section 5.2.6, "Print Pages" as well as categorized as described in Section 5.2.7, "Categorization".

6.5.1. Creating a System under Test

New systems under test may be created by clicking on the **New** button (Figure 6.18). The system under test ID is automatically assigned by Klaros-Testmanagement and the version can be specified by the user. The system under test is created in the currently selected project.

Clicking on the delete icon removes the system under test.

Why is the Delete button disabled?

Systems under test which are already referenced by test runs cannot be deleted, so their delete icon is disabled ( ).

Clicking on the edit icon displays the details page of the system under test.

6.5.2. Bulk Actions

Feature only available in Klaros-Testmanagement Enterprise Edition

Some bulk actions are available on the *Systems under Test* page. Bulk actions are described in Section 5.2.10, "Bulk Actions".

The bulk actions available on this page are: *Edit*, *Print*, *Delete* and *Assign to category*. The bulk action buttons can be found above the table on the left.

6.5.3. System under Test Details

This section allows in-depth editing of an individual system under test, including user defined properties. The executed test results and change history for the system under test are also visible in this section.

6.5.3.1. System under Test Overview

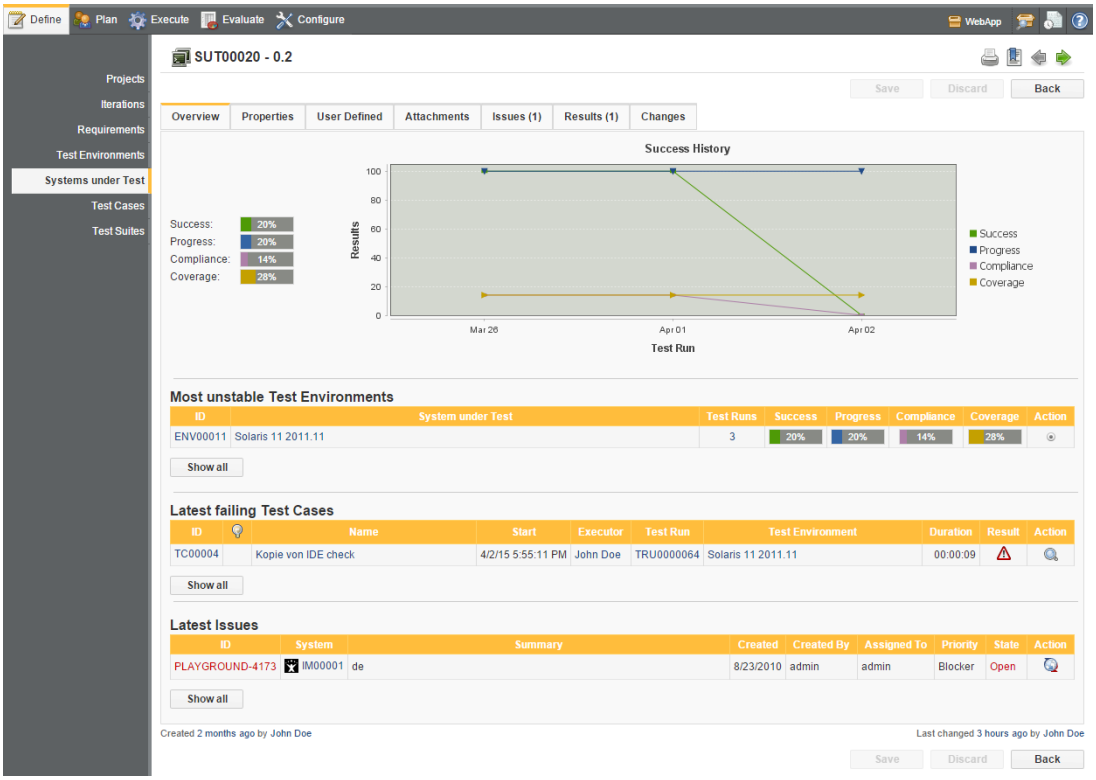


Figure 6.19. The System under Test Overview Tab

- *Success*

  The success rate shows how many test cases have been successfully executed in this system under test.

- *Progress*

  The progress rate shows how many test cases have been executed at least once in this system under test.

Note

In contrast to the success rate, the coverage rate considers only if a test case has been executed at least once in this test environment, regardless of the result (passed, failed, error).

- *Compliance*

  The compliance rate shows how many test cases, which are part of this iterations requirements, have been successfully executed in this system under test.

- *Coverage*

  The coverage rate shows how many test cases, which are part of a requirement, have been executed at least once in this system under test.

Note

In contrast to the compliance rate, the coverage rate considers only if a test case has been executed at least once in the iteration, regardless of the result (passed, failed, error).

- *Success History*

  This graph shows changes in the compliance, coverage, progress and success rates of this test environment over time. The timeline is automatically adjusted depending on the given timeframe.

- *Most unstable Test Environments*

  This table displays the test environments which have the lowest success and progress rates in conjunction with this system under test.

Note

This table is limited to five entries. The other test environments can be viewed by pressing the *Show All* button.

- *Latest failing Test Cases*

  This table lists the test cases which recently have been failed during a test run for this system under test alongside with the appertaining test case result.

Note

This table is limited to five entries. The other system under tests can be viewed by pressing the *Show All* button.

- *Latest Issues*

  This list shows the latest issues which have been created in conjunction with this system under test during test execution or linked to this system under test after its creation.

Note

This table is limited to five entries. The other system under tests can be viewed by pressing the *Show All* button.

6.5.3.2. User Defined Properties

Feature only available in Klaros-Testmanagement Enterprise Edition

User defined properties can be used to customize systems under test within Klaros-Testmanagement by adding custom fields. This process is described in Section 5.2.11, “User Defined Properties”.

6.5.3.3. Attachments

Attachments may be added to systems under test. This process is described in Section 5.2.12, “Attachments”.

6.5.3.4. Issues

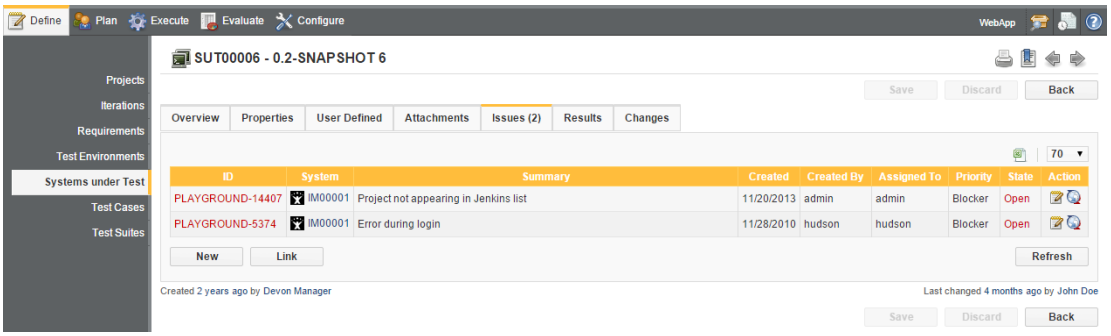


Figure 6.20. The Test Case Issues Screen

This tab lists all issues that have been linked to this system under test. Pressing the **New** button opens the *Issue Details* page (Section 9.7.2, “Issue Details (Creating a new Issue)”), where new issues can be created and linked to this system under test. By pressing the **Link** button it is possible to link existing issues to this system under test (Section 9.7.3, “Link Issues”).

6.5.3.5. Results

The result tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab showing the test results related to this system under test as described in Section 5.2.14, “Related Test Runs and Results”.

6.5.3.6. Bookmarking Systems under Test

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

Systems under test may be linked to from outside of Klaros-Testmanagement using the link on the bookmark icon. Right-click the icon and choose *Copy Link Location* (Firefox/Chrome) or the equivalent option.

6.6. Test Cases

A test case is a set of input values, execution preconditions, expected results and execution postconditions developed for a particular objective or test condition in order to determine whether or not an application or software system meets its specifications.

Test cases may be created, removed, duplicated, updated and assigned to categories on this page. The displayed test cases may be filtered and sorted as shown in Section 5.2.5, "Filtering and Sorting" (Figure 6.21).

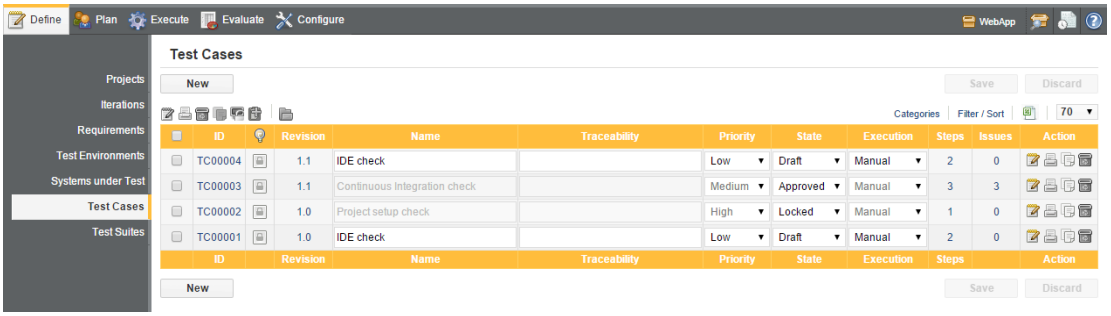


Figure 6.21. The Test Cases Screen

The *Test Cases* screen shows a table with all of the test cases defined for the current project. Each row of the table represents a test case, the columns represent attributes of the test case. Clicking on the fields of the table allows some of the most important values of a test case (*Name*, *Traceability*, *Priority*, *State* and *Execution*) to be directly edited. More attributes of a test case and its steps can be edited in on the test case details page by clicking on the Edit icon (Section 6.6.3, "Test Case Details").

The leftmost column allows test cases to be selected for bulk actions. The uppermost checkbox is a global select button, which selects or deselects all elements shown in the table. Please note that the displayed elements are subject to the current search and catagorization criteria.

Test cases may be viewed on a printer-friendly page as described in Section 5.2.6, "Print Pages" as well as categorized as described in Section 5.2.7, "Categorization".

Warning Sign in the ID Column

A test case which has execution set to manual and contains no steps is not executable. To indicate this, a warning sign ( ) is displayed in the ID column.

Clicking on the delete icon removes the test case.

Warning

Only test cases that are not referenced by results or test suites can be deleted.

6.6.1. Bulk Actions

Feature only available in Klaros-Testmanagement Enterprise Edition

Some bulk actions are available on the *Test Cases* page. Bulk actions are described in Section 5.2.10, "Bulk Actions".

The bulk actions available on this page are: *Edit*, *Print*, *Delete*, *Duplicate*, *Create new Revisions*, *Create new Test Suite* and *Assign to category*. The bulk action buttons can be found above the table on the left.

6.6.2. Creating Test Cases

Clicking the **New** button creates a new test case (Figure 6.21). An empty row is added at the beginning of the table. The fields *Name*, *Traceability*, *Priority*, *State* and *Execution* of the test case can be specified. The test case ID is automatically assigned by Klaros-Testmanagement. The test case is created in the currently selected project. Clicking the edit icon opens the test case details page where more properties of the test case can be set (Section 6.6.3, "Test Case Details"). Clicking the duplicate icon allows the user to create a copy of the test case.

6.6.3. Test Case Details

6.6.3.1. Test Case Overview

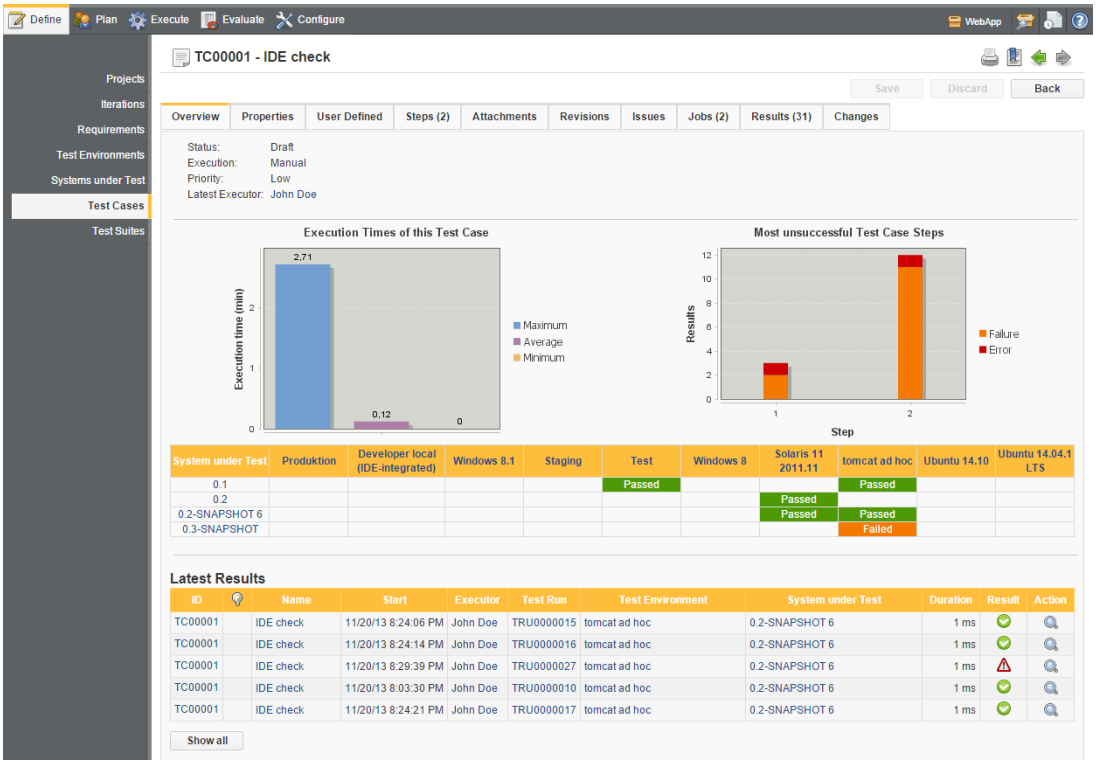


Figure 6.22. The Test Case Overview Tab

- *Execution Times of this Test Case*

  This graph shows the minimum, maximum and average execution times for this test case.

Tip

This can be very useful in determining the estimated time needed for future executions of this test case.

- *Most unsuccessful Test Case Steps*

  This graph shows all test case steps of this test case and which steps have been marked as failed or error during a test run.

  Tip

  This graph can be used for determining possible misconfigurations, if errors happen more frequently at one specific step for example.

- *System under Test Overview*

  This table shows the latest result of this test case for every system under test and test environment this test case has been executed in. Pressing the result will open up the appertaining test case result.

- *Latest Results*

  This table shows the latest results for this test case, where the latest result is shown on the top.

  Note

  This table is limited to five entries. The other issues can be viewed by pressing the *Show All* button.

- *Latest Issues*

  This list shows the latest issues which have been created in conjunction with this test case during test execution or linked to this test case after its creation.

  Note

  This table is limited to five entries. The other issues can be viewed by pressing the *Show All* button.

6.6.3.2. Test Case Properties

This page (Figure 6.23) allows the user to view or change the following attributes of the selected test case:

- *ID*

  Assigned automatically.

- *Revision*

  Assigned automatically and incremented every time a test case is changed.

- *Name*

The editable name of the test case.

- *Description*

  The textual description of the test case.

- *Precondition*

  The precondition for this test case, describing the requirements for executing the test case, e.g. "It is required that the database is initialized."

- *Postcondition*

  The postcondition defines the criteria that must be fulfilled after the test case has been executed, e.g. the postcondition of a successful login test case is that the user is authenticated and the home page of the application appears.

- *Area*

  The area this test case covers (*Functional*, *Non Functional*, *Structural*, *Regression*, *Retest*).

- *Design Technique*

  The test design view on the system under test: Black-Box or White-Box.

- *Variety*

  The result expectation of the test: Whether a positive or negative result is expected.

- *Team*

  The team that is responsible for the test case.

- *Level*

  The test level of the test case, for example unit, integration or system test.

- *Priority*

  The priority of the test case: *Low*, *Medium* or *High*.

- *Docbase*

  A reference to the document this test case is based on, which may e.g. contain the requirements this test case is related to.

- *Note*

  A note that may contain additional information regarding the test case that may be useful for users to know.

- *Dependency*

  The dependencies of the test case (it may depend on another test case).

- *Evaluation*

The mode of test result evaluation: *Manual* or *Automated*.

- *Execution*

  The mode of test case execution: *Manual* or *Automated*.

- *Traceability*

  A reference to the corresponding requirement, use case or work package.

- *State*

  The test case state determines whether it is editable, executable or visible in result listings. Only test cases with the state *Draft* are editable. A test case with the state *Locked* is not executable and a test case with the state *Skip* is not shown in any reports.

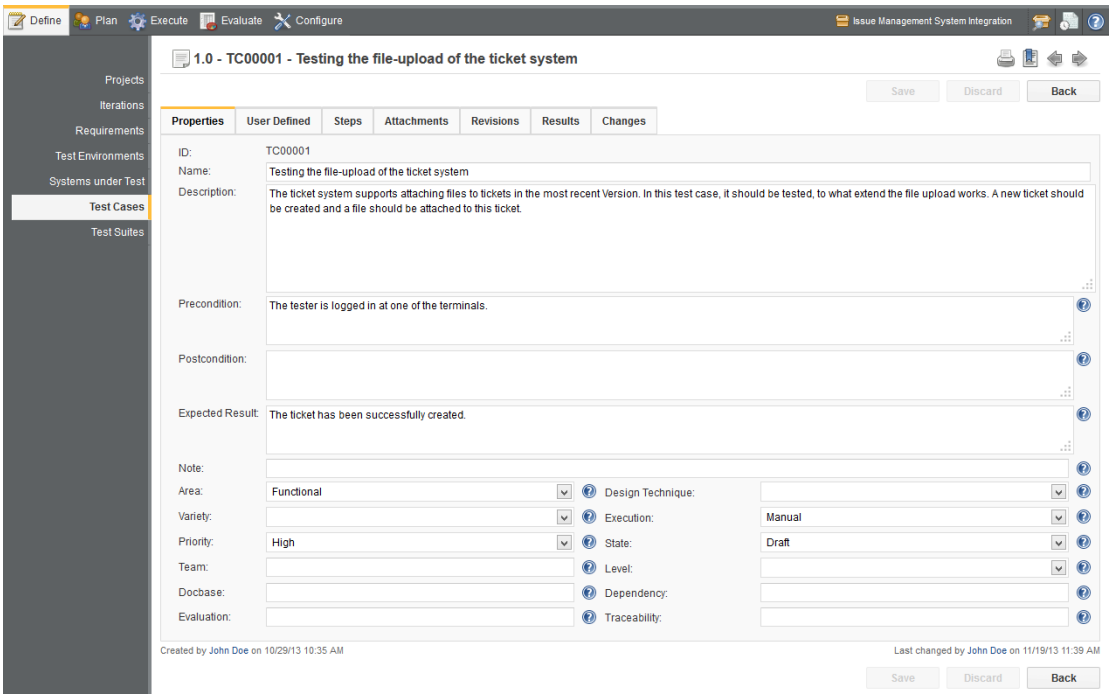


Figure 6.23. The Edit Test Cases Screen

How to create a new revision of a test case?

Clicking the *New Revision* button creates a new revision of the test case. This button is disabled if there are any unsaved changes to the test case.

6.6.3.3. User Defined Properties

Feature only available in Klaros-Testmanagement Enterprise Edition

User defined properties can be used to customize test cases within Klaros-Testmanagement by adding custom fields. This process is described in Section 5.2.11, “User Defined Properties”.

6.6.3.4. Edit Test Steps

In the *Steps* tab, Test steps may be added to or removed from the test case. The following attributes of single test steps may also be changed:

- *Description*

  The description of the action to take, e.g. `Enter name and password, and click the Login button.`

- *Expected Result*

  The (visible) behavior of the system predicted while carrying out the test step e.g. `The user recieves a message that the login succeeded.`

- *Precondition*

  Condition that must be fulfilled before the execution of the test step, e.g. `The user is not yet logged in.`

- *Postcondition*

  The environmental and state conditions that must be fulfilled after the execution of the test case e.g. `The user is authenticated and has access to the system.`

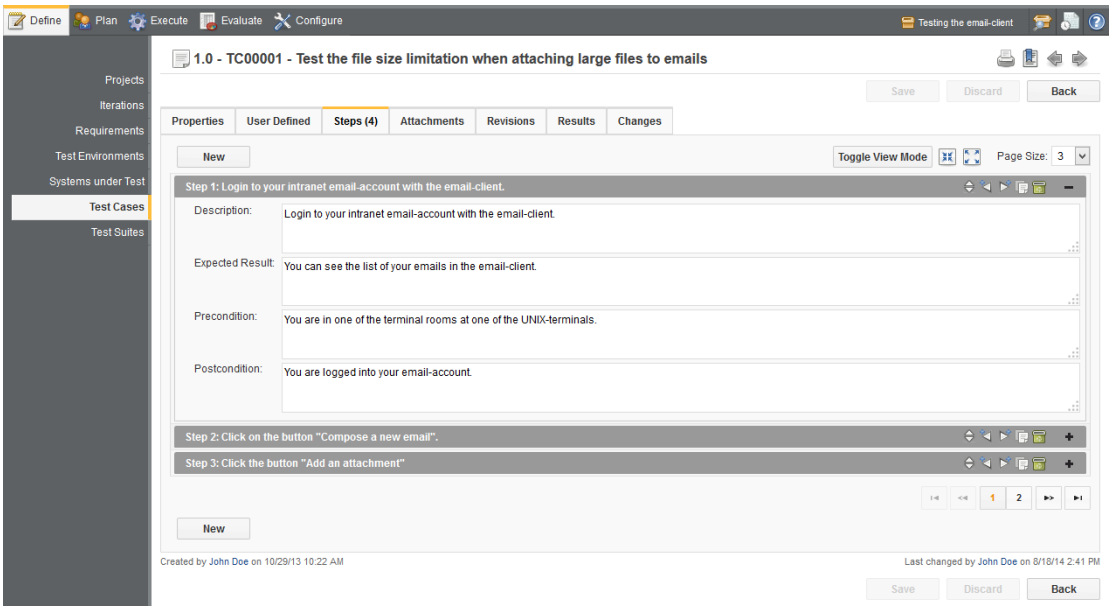


Figure 6.24. The Edit Test Steps Screen

To change the order of the test steps the icon is used. This opens a dialog which allows the test step to be inserted in another position in the list. To add a new test step to the end of the list, click the icon. Test steps are inserted before or after a given step using the and icons. The clone icon is used to duplicate a step at another position. Individual steps may be minimized while editing to help to keep track of Steps which are being edited.

Pressing the **Toggle View Mode** displays the individual test case steps in a tabbed view to save space. The and icons can be used to expand or collapse all test case steps.

Test steps are removed by pressing on the delete icon. The *Save* button must be pressed to persist this change. (Figure 6.23).

6.6.3.5. Attachments

Attachments may be added to test cases. This process is described in Section 5.2.12, "Attachments".

6.6.3.6. Revisions

Test cases are revisionable. The process of revisioning an object is described in Section 5.2.13, "Revisions".

6.6.3.7. Issues

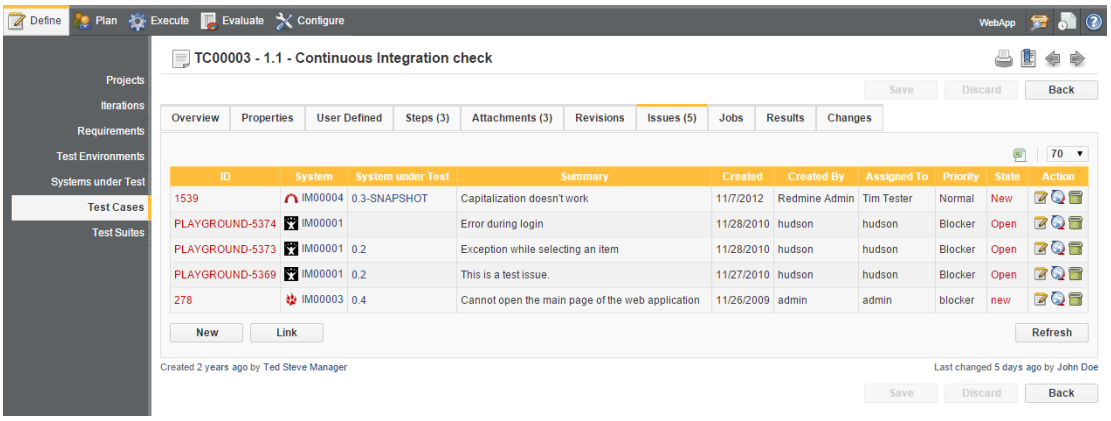


Figure 6.25. The Test Case Issues Screen

This tab lists all issues that have been linked to this test case. Pressing the **New** button opens the *Issue Details* page (Section 9.7.2, "Issue Details (Creating a new Issue)"), where new issues can be created and linked to this test case. By pressing the **Link** button it is possible to link existing issues to this test case (Section 9.7.3, "Link Issues").

6.6.3.8. Jobs

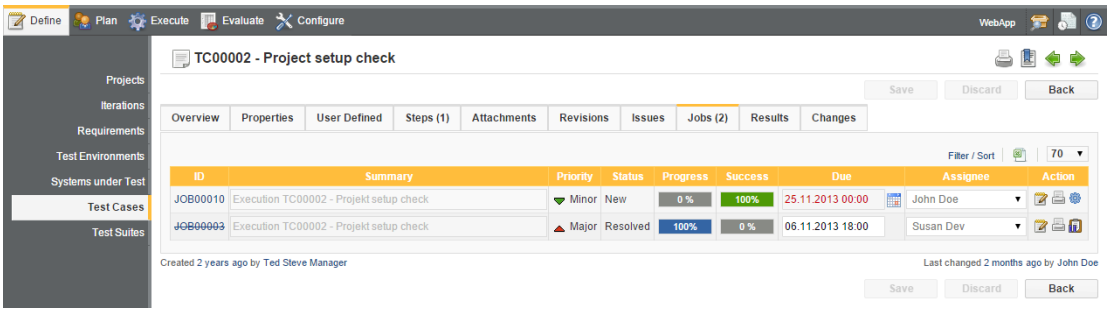


Figure 6.26. The Test Case Jobs Screen

This tab lists all jobs in which this testcase is included. These jobs can be directly executed from within this tab. See Section 7.1, "Maintain Jobs" for more information on how to manage jobs.

6.6.3.9. Results

The result tab is further divided into a *Test Runs* and *Test Case Results* tab showing the test results related to this test case as described in Section 5.2.14, “Related Test Runs and Results”.

6.6.3.10. Bookmarking Test Cases

Feature only available in Klaros-Testmanagement Enterprise Edition

Test cases may be linked to from outside of Klaros-Testmanagement using the link on the bookmark icon. Right-click the icon and choose *Copy Link Location* (Firefox/Chrome) or the equivalent option.

6.7. Test Suites

A test suite is a set of test cases which can be executed sequentially. The results of the test suite execution are grouped together in Klaros-Testmanagement and can be used to verify and ensure that a system meets its design specifications and requirements.

Test suites may be created, removed, duplicated, updated and assigned to categories on this page. The displayed test suites may be filtered and sorted as described in Section 5.2.5, “Filtering and Sorting” (Figure 6.27).

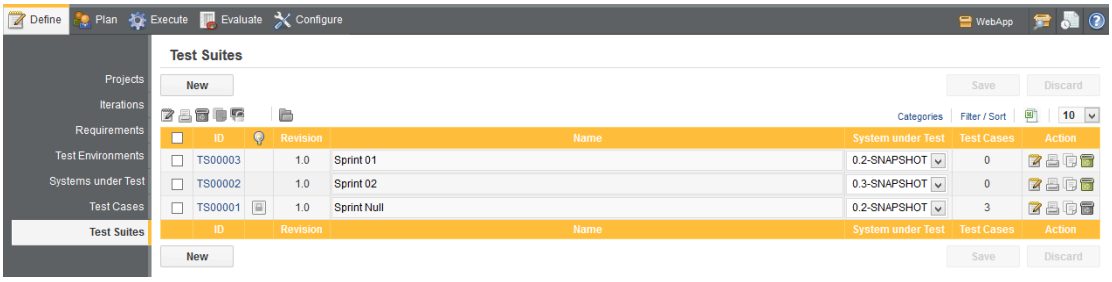


Figure 6.27. The Test Suites Screen

The *Test Suites* screen shows all of the test suites in the current project. Direct editing of the displayed attributes is possible here.

The leftmost column allows test suites to be selected for bulk actions. The uppermost checkbox is a global select button, which selects or deselects all elements shown in the table. Please note that the displayed elements are subject to the current search and catagorization criteria.

Test Suites may be viewed on a printer-friendly page as described in Section 5.2.6, “Print Pages” as well as categorized as described in Section 5.2.7, “Categorization”.

6.7.1. Bulk Actions

Feature only available in Klaros-Testmanagement Enterprise Edition

Some bulk actions are available on the *Test Suites* page. Bulk actions are described in Section 5.2.10, “Bulk Actions”.

The bulk actions available on this page are: *Edit*, *Print*, *Delete*, *Duplicate selected Items*, *Create new Revisions*, *Add Test Cases* and *Assign to Category*. The bulk action buttons can be found above the table on the left.

6.7.2. Create Test Suites

New test suites can be created by clicking the **New** button (Figure 6.27). The test suite ID is automatically assigned and the *Description* and *System under Test* may be specified by the user. The test suite is created in the currently selected project. Clicking the Edit icon opens the *Test Suite Details* section (Section 6.7.3, “Test Suite Details”). Clicking the duplicates a test suite.

6.7.3. Test Suite Details

Clicking the Edit icon opens the *Test Suite Details* page and allows editing of the test suite's description and its system under test and of the list of the test cases in this test suite, also changing their order (Figure 6.28).

The assigned System under Test of a Test Suite

The assignment of a test suite with a system under test does not affect the functionality of Klaros-Testmanagement. The test suites can be still run with other systems under test. This attribute is just a guideline for the user.

Clicking the icon adds the test case to the test suite, clicking the icon removes the test case from the test suite and clicking the opens a dialog which allows the user to change the order of the test cases.

The icon allows to quickly find and remove all duplicate test case entries in the test suite. After invocation it displays a dialog listing all duplicates found. If the dialog is confirmed all duplicates are removed from the test suite and the test suite is saved.

How to create a new revision of a test suite?

Clicking the *New Revision* button creates a new revision of the test suite. This button is disabled if there are any unsaved changes to the test suite.

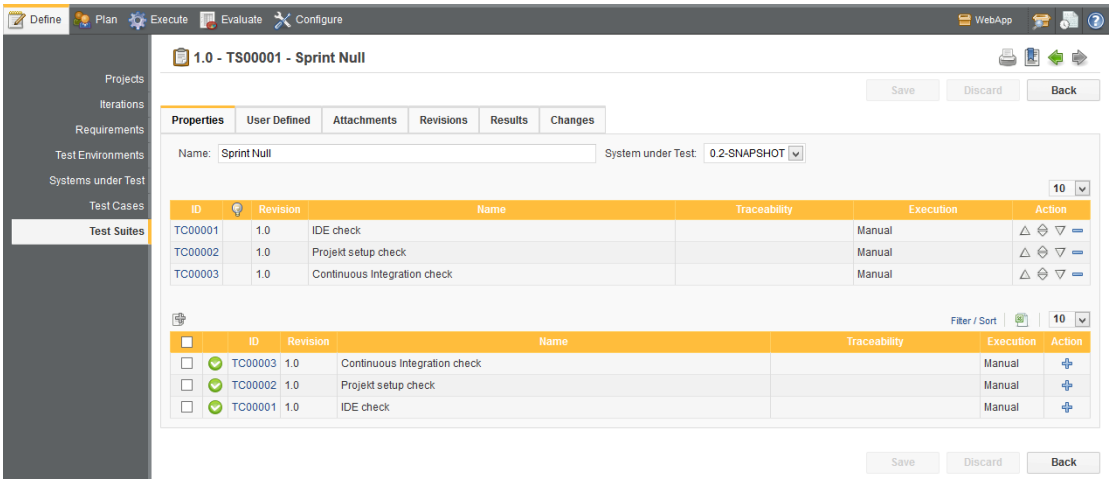


Figure 6.28. The Test Suite Details Screen

Warning sign in the ID column

A manual test case which contains no steps is not executable. To indicate this, a warning sign (⚠) is displayed in the ID column.

6.7.3.1. Test Suite Overview

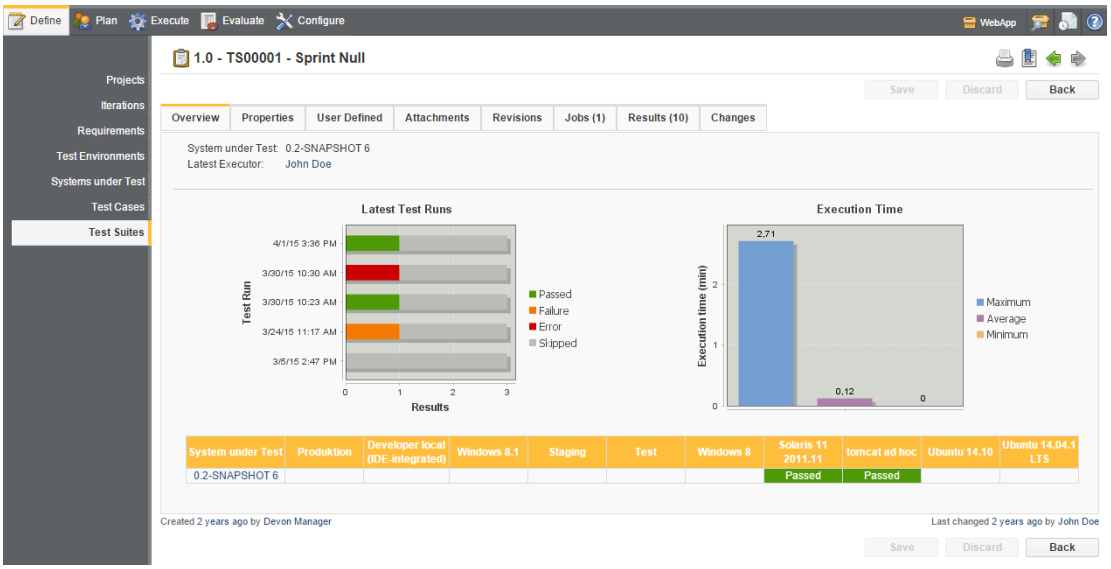


Figure 6.29. The Test Suite Overview Tab

- *Latest Test Runs*

  This graph shows the number of passed, failed, erroneous and skipped test cases for the latest five test runs for this test suite.

- *Execution Times of this Test Suite*

  This graph shows the minimum, maximum and average execution times for this test suite.

Tip

This can be very useful in determining the estimated time needed for future executions of this test suite.

- *System under Test Overview*

  This table shows the latest result of this test suite for every system under test and test environment this test suite has been executed in. Hovering the mouse cursor over a single result will sho a more detailed overview of the individual test case results of this test suite. Pressing the result will open up the appertaining test case result.

6.7.3.2. Test Suite Properties

6.7.3.2.1. User Defined Properties

Feature only available in Klaros-Testmanagement Enterprise Edition

User defined properties can be used to customize test suites within Klaros-Testmanagement by adding custom fields. This process is described in Section 5.2.11, “User Defined Properties”.

6.7.3.2.2. Attachments

Attachments may be added to test suites. This process is described in Section 5.2.12, “Attachments”.

6.7.3.2.3. Revisions

Test suites are revisionable. The process of revisioning an object is described in Section 5.2.13, “Revisions”.

6.7.3.2.4. Jobs

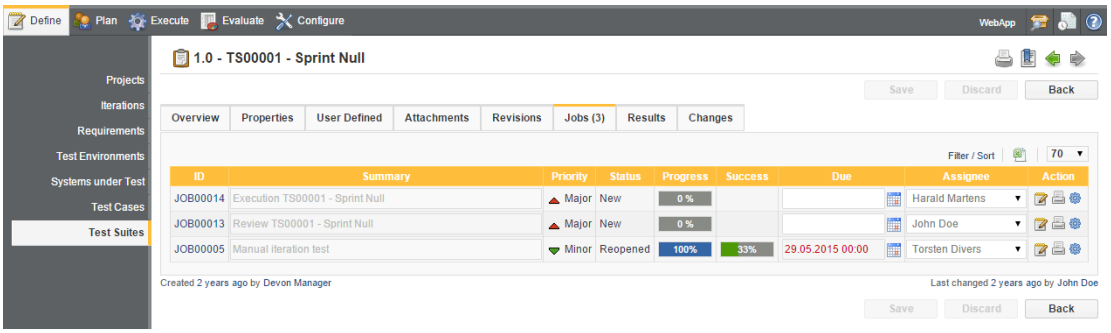


Figure 6.30. The Test Suite Jobs Screen

This tab lists all jobs in which this testsuite is included. These jobs can be directly executed from within this tab. See Section 7.1, “Maintain Jobs” for more information on how to manage jobs.

6.7.3.2.5. Results

The result tab is further divided into a *Test Runs* and *Test Suite Results* tab showing the test results related to this test suite as described in Section 5.2.14, “Related Test Runs and Results”.

6.7.3.2.6. Bookmarking Test Suites

Feature only available in Klaros-Testmanagement Enterprise Edition

Test suites may be linked to from outside of Klaros-Testmanagement using the link on the bookmark icon. Right-click the icon and choose *Copy Link Location* (Firefox/Chrome) or the equivalent option.

Chapter 7. Plan

This section describes the plan section of Klaros-Testmanagement. The plan section is used to define and maintain jobs, generate jobs from test cases and test suites, and to track the workload and progress of users in the selected project.

Feature only available in Klaros-Testmanagement Enterprise Edition

7.1. Maintain Jobs

This section is used to define and maintain jobs. Jobs may consist of the execution of test cases or test suites or any other possible task. Jobs can be nested and assigned to individual users. The executions and results of jobs are automatically tracked by Klaros-Testmanagement.

Jobs can be created, removed, updated and nested on this page (Figure 7.1). The displayed Jobs may be filtered and sorted as described in Section 5.2.5, “Filtering and Sorting”.

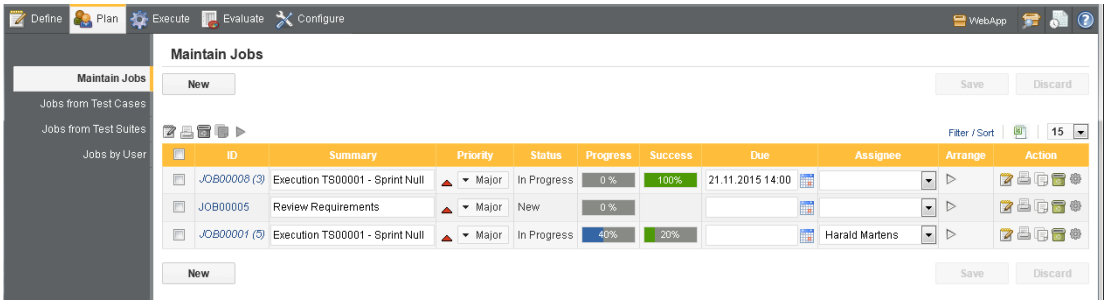


Figure 7.1. The Maintain Jobs Screen

The *Maintain Jobs* page shows all of the jobs defined for the current project. Clicking on the table fields allows direct editing of the attributes.

The leftmost column allows jobs to be selected and bulk actions to be carried out. The uppermost checkbox is a global select button, which allows bulk actions to be carried out for all jobs that match the current search criteria.

The *Action* column contains buttons for carrying out actions on individual jobs. The icon opens the *Job Details Page* for the corresponding job (described in Section 7.1.5, “Job Details”). The duplicate icon duplicates the corresponding job, optionally including its sub-jobs. The delete icon removes the corresponding job. The icon executes the corresponding job (described in Section 7.1.4, “Executing a Job”). The icon imports test results if the job references an automated test case.

Jobs may be viewed on a printer-friendly page as described in Section 5.2.6, “Print Pages”.

7.1.1. Bulk Actions

Some bulk actions are available on the *Maintain Jobs* page. Bulk actions are described in Section 5.2.10, “Bulk Actions”.

The bulk actions available on this page are: *Edit*, *Delete*, *Duplicate*, *Assign to Parent* and, if viewing sub-jobs, *Move to new Position* and *Move up one Level*. The bulk action buttons can be found above the table on the left.

7.1.2. Creating Jobs

Clicking the **New** button creates a new job in the selected project (Figure 7.1). The job ID is automatically assigned by Klaros-Testmanagement. The *Summary*, *Priority*, *Due Date* and *Assignee* fields can be specified here. The new job must be persisted by clicking the *Save* button.

7.1.3. Arranging Jobs

Jobs can be arranged using the arrow icons in the *Arrange* column. The icon opens a dialog which allows the user to move the job to a new position, and the and icons change the job hierarchy. Using the icon will put the job in the same hierarchy level as its parent job. Using the opens a dialog which allows the user to choose a new parent job from the job's siblings.

Parent Jobs have specific features:

- A parent job can not be assigned to any test case or test suite
- A parent job can also contain multiple parents with subordinated jobs
- If the arrow key on a parent is gray, this means there is no other job parent available

To subordinate a job, create a job of type *Task* that does not have a test case or test suite. Then, select a job that should be subordinated by clicking the icon (this job can have a test case or test suite). Here you can choose witch job should be the parent, then click **Assign**. Next to the job ID the number of subordinate jobs in this parent is shown. The success- and progress rate shown on the parent job is derived from the contained jobs.

Warning

Jobs which have a test case or test suite assigned to them may not have sub-jobs, and so will not be available as possible parent jobs in the *New Parent Dialog*.

7.1.4. Executing a Job

Clicking the icon executes the job. A user in the role tester is only able to execute or jobs which are assigned to him.

If the job is associated with a test case, the *Run Single Test Case* dialog will be opened. A detailed description of how to run a test case can be found in Section 8.2, “Run Test Case”.

If the job is associated with a test suite, the *Run Test Suite* dialog will be opened. A detailed description of how to run a test suite can be found in Section 8.3, “Run Test Suite”.

Note

If the job is associated with an automated test case, or the test suite contains only automated test cases, Klaros-Testmanagement will display the page to import test results. More information about importing results can be found in Section 8.5, “Import Test Results”.

7.1.5. Job Details

7.1.5.1. Job Overview

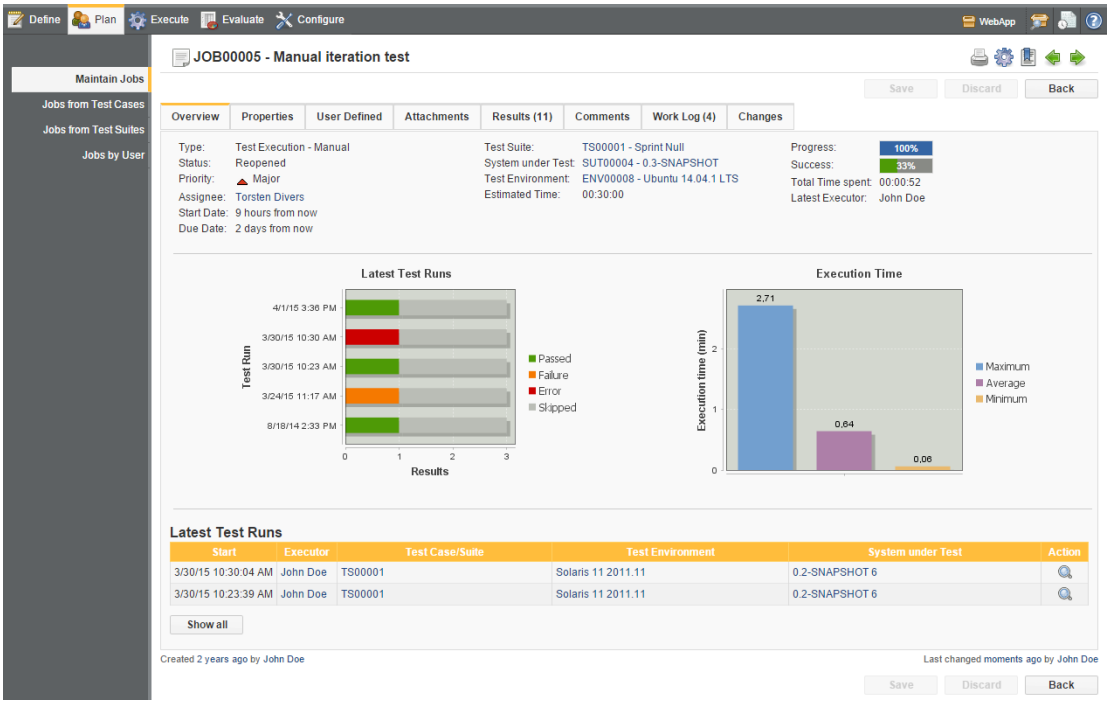


Figure 7.2. The Job Overview Tab

- *Type*

  The job type (*Test Execution - Automated*, *Test Execution - Manual* or *Task* (if the job isn't related to a test case)).

- *Status*

  The status of the job.

- *Priority*

  The priority of the job.

- *Assignee*

  Who is assigned for this job.

- *Start Date*

  The earliest point in time this job can be executed by a tester.

- *Due Date*

  The point in time where the execution of the job has to be completed at the latest.

- *Test Case / Test Suite*

  The test case or test suite which is covered by this job.

- *System under Test*

  The system under test which has to be used when executing this job.

- *Test Environment*

  The test environment which has to be used when executing this job.

- *Estimated Time*

  The estimated time needed to complete this job.

- *Progress*

  The progress rate shows how many test cases which are covered by this job have been executed, regardless the result.

- *Success*

  The success rate shows how many of the test cases which are covered by this job have been successfully executed in the latest test run.

- *Total Time Spent*

  How much time has actually been spent testing. This value is calculated by adding up the individual execution times from the work log.

- *Latest Executor*

  The executor which has executed the latest test run of this job.

- *Latest Test Runs*

  This graph displays the accumulated test case results of the latest test runs for this job in descending order.

- *Executon Time*

  This graph shows the minimum, average and maximum execution times for this job.

- *Latest Test Runs*

  This table shows the latest completed test runs for this job.

Note

By default, this table shows only five entries. Pressing the *Show all* button opens up the results tab in which all test runs are listed.

7.1.5.2. Job Properties

This section allows in-depth editing of a single job. the executed test results and change history for the Job are also shown in this section. (Figure 7.4). The *Job Properties page* allows the user to change the following attributes:

- *Name*

  The name or summary of the Job

- *Description*

  The description of the Job

- *Priority*

  The priority of the job. Possible values are, in order of severity, *Trivial*, *Minor*, *Major*, *Critical* and *Blocker*. Blocker means that the job should be executed before further progress in the project can be made.

- *Status*

  The status of the job. Possible values are *New*, *In Progress*, *Resolved*, *Closed*, *Reopened* and *Rejected*.

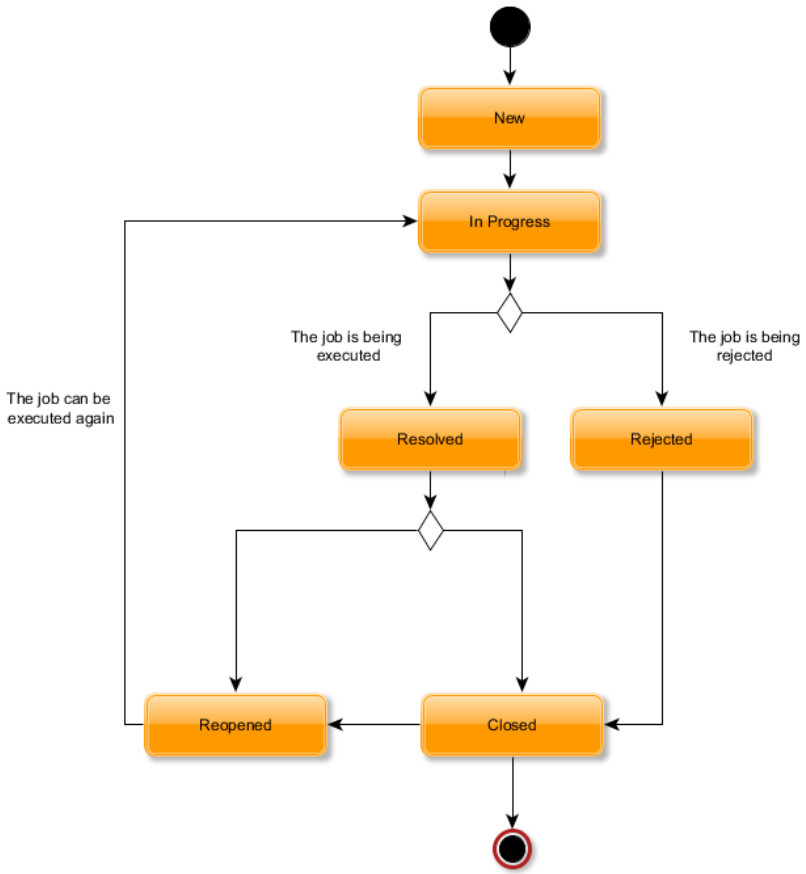


Figure 7.3. The Job States

- *Estimated Time*

The estimated time in hours that the job will take.

- *Finished*

  The current percentage of completion of the job.

- *Start Date*

  The date when the job should be started.

- *Due Date*

  The deadline when the job should be finished.

- *Assignee*

  The user responsible for the job.

- *Test Case*

  The test case that should be executed with this job. Either a test case or a test suite can be assigned to a job. This field is unavailable if the job contains sub-jobs.

- *Test Suite*

  The test suite that should be executed with this job. Either a test case or a test suite can be assigned to a job. This field is unavailable if the job contains sub-jobs.

- *System under Test*

  The system under test for which this job should be executed.

- *Test Environment*

  The test environment in which the job should be executed.

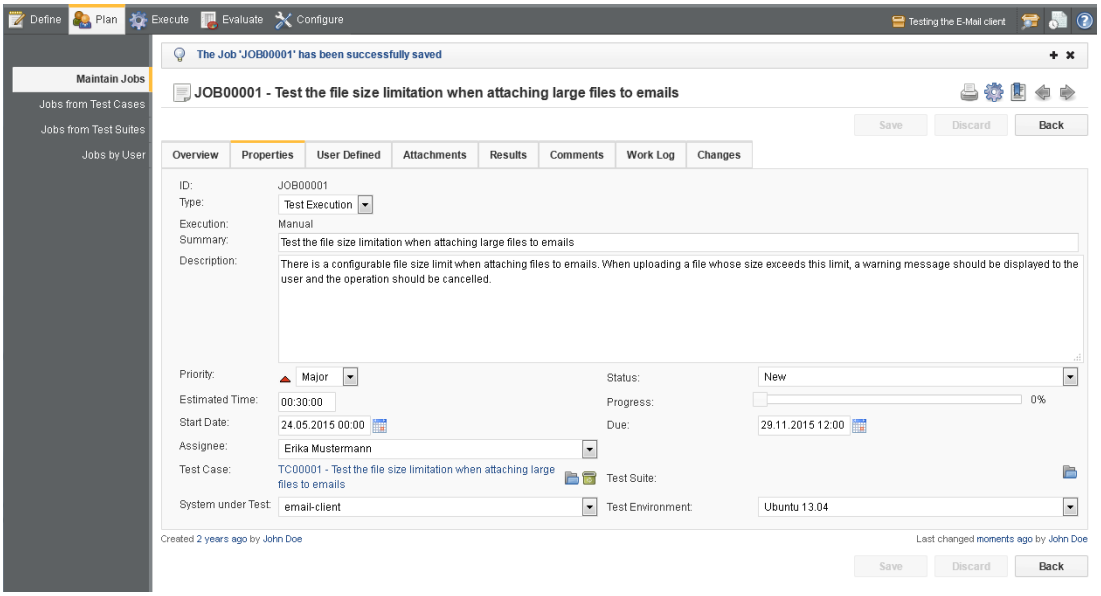


Figure 7.4. The Job Details Screen

It is possible to assign a test case or test suite to a job, but only if the job does not contain sub-jobs. Clicking the icon opens a dialog in which a test case or a test suite can be selected. Only one item can be selected at the same time. Clicking the icon removes the element from the job.

This section also contains a *Test Runs* tab and a *Results* tab as described in Section 5.2.14, "Related Test Runs and Results".

7.1.5.2.1. User Defined Properties

User defined properties can be used to customize jobs within Klaros-Testmanagement by adding custom fields. This process is described in Section 5.2.11, "User Defined Properties".

7.1.5.2.2. Attachments

Attachments may be added to jobs. this process is described in Section 5.2.12, "Attachments".

7.1.5.2.3. Results

The result tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab showing the test results related to this job as described in Section 5.2.14, "Related Test Runs and Results".

7.1.5.2.4. Comments

This tab allows a user to add comments by clicking on the **Add Comment** button.

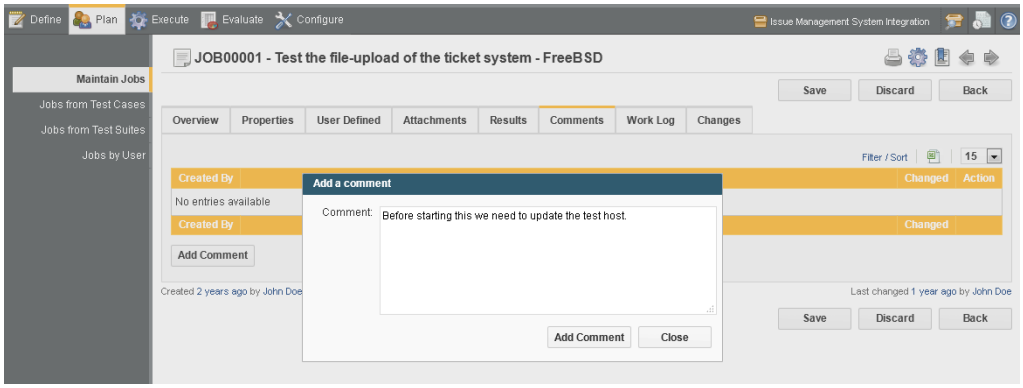


Figure 7.5. The Add a Comment Dialog

The text entered in this dialog will be saved as a comment on the job once the **Add Comment** button is clicked and saved. To modify the comment, use the icon, to remove it, click the icon.

7.1.5.2.5. Work Log

Every executed test run related to this job is logged in the *Work log* tab. It is also possible to add own work logs to the job. When clicking the **Log Work** button a dialog is shown where all the information can be set.

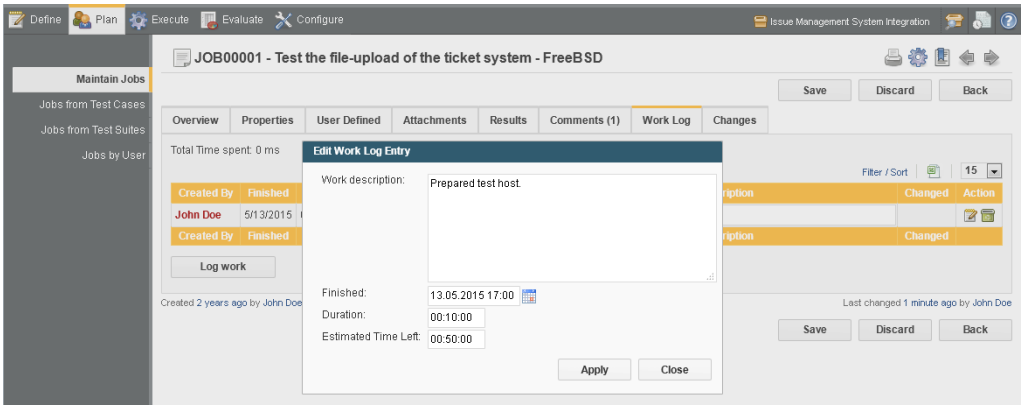


Figure 7.6. The Log Work Dialog

The duration field must contain a number greater than zero in order for the work unit to be logged. Then click **Add** and then **Save**, to save you log work. To modify a work log entry, use the icon, to remove it, click the icon.

7.1.5.2.6. Bookmarking Jobs

Jobs may be linked to from outside of Klaros-Testmanagement using the link on the bookmark icon. Right-click the icon and choose *Copy Link Location* (Firefox/Chrome) or the equivalent option.

7.1.6. Jobs from Test Cases

On this page, the user can generate jobs from the selected test cases.

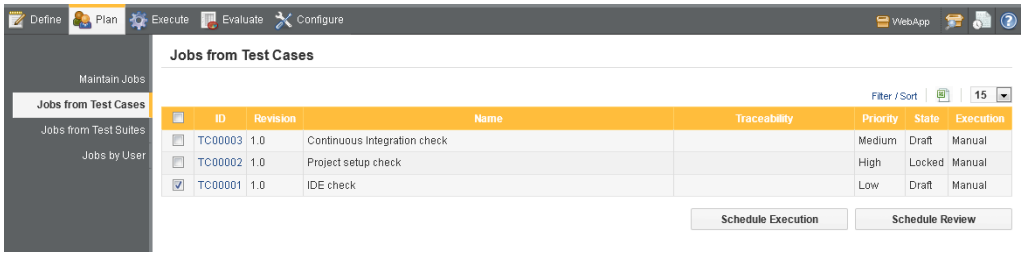


Figure 7.7. The Jobs from Test Cases Screen

When the **Generate** button is clicked, Klaros-Testmanagement generates a job for every selected test case using the specified *Assignee*, *System under Test* and *Test Environment*.

Each generated job is created with the corresponding test case set in its *Test Case* field.

7.1.7. Jobs from Test Suites

On this page, the user can generate jobs from the selected test suites.

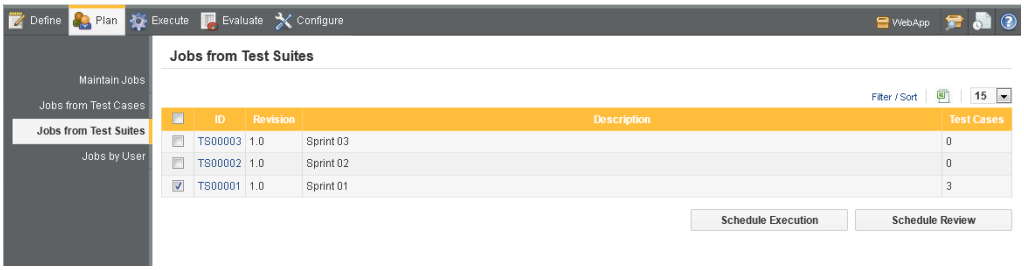


Figure 7.8. The Jobs from Test Suites Screen

When the **Generate** button is clicked, Klaros-Testmanagement generates a job for every selected test suite using the specified *Assignee*, *System under Test* and *Test Environment*.

Each generated job is created with the corresponding test suite set in its *Test Suite* field.

7.1.8. Jobs by User

This section is used to track the workload and progress of individual users in the selected project.

On the *Jobs by User* page the workload, work duration and progress of every user for the current active project is shown. Clicking the icon displays the details page (Figure 7.10) for the selected user.

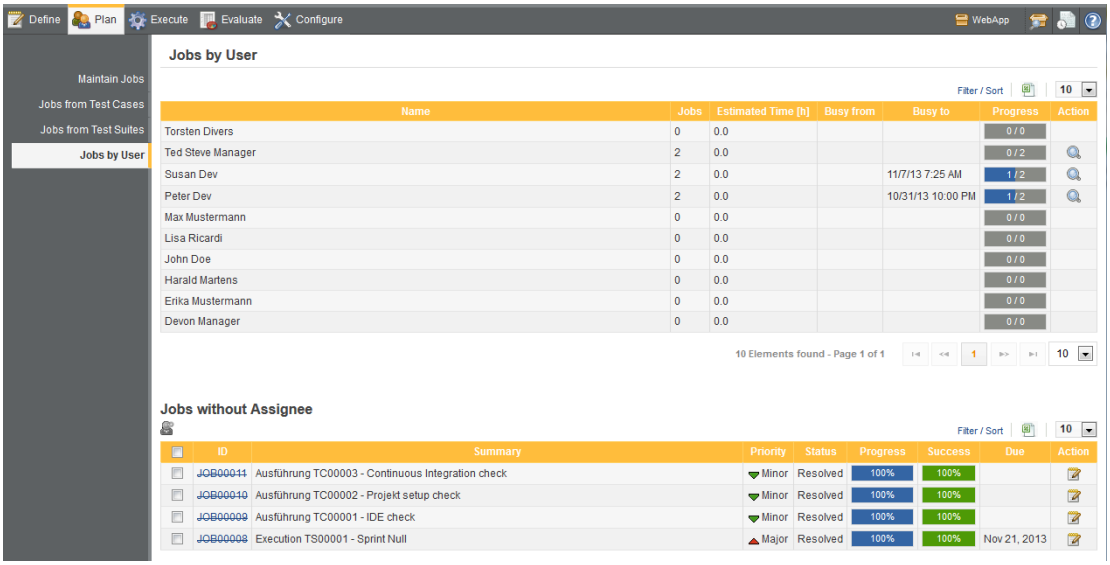

Define
Plan
Execute
Evaluate
Configure
WebApp
Maintain Jobs
Jobs from Test Cases
Jobs from Test Suites
Jobs by User
Jobs by User
Filter / Sort
10
Name
Jobs
Estimated Time [h]
Busy from
Busy to
Progress
Action
Torsten Divers 0 0.0 0 / 0
Ted Steve Manager 2 0.0 0 / 2
Susan Dev 2 0.0 11/7/13 7:25 AM 1 / 2
Peter Dev 2 0.0 10/31/13 10:00 PM 1 / 2
Max Mustermann 0 0.0 0 / 0
Lisa Ricardi 0 0.0 0 / 0
John Doe 0 0.0 0 / 0
Harald Martens 0 0.0 0 / 0
Erika Mustermann 0 0.0 0 / 0
Devon Manager 0 0.0 0 / 0
10 Elements found - Page 1 of 1
Jobs without Assignee
Filter / Sort
10
ID
Summary
Priority
Status
Progress
Success
Due
Action
JOB00011 Ausführung TC00003 - Continuous Integration check Minor Resolved 100% 100%
JOB00010 Ausführung TC00002 - Projekt setup check Minor Resolved 100% 100%
JOB00009 Ausführung TC00001 - IDE check Minor Resolved 100% 100%
JOB00008 Execution TS00001 - Sprint Null Major Resolved 100% 100% Nov 21, 2013

Figure 7.9. The Jobs by User Screen

7.1.8.1. Jobs by User - Details

This page displays the workload, work duration and progress for a user in the currently selected project. The user's jobs are shown in a table, and a Gannt chart aids visualization of the user's workload. Clicking the edit icon of a job opens the job details page for that job. Further information can be found in Section 7.1.5, “Job Details”.

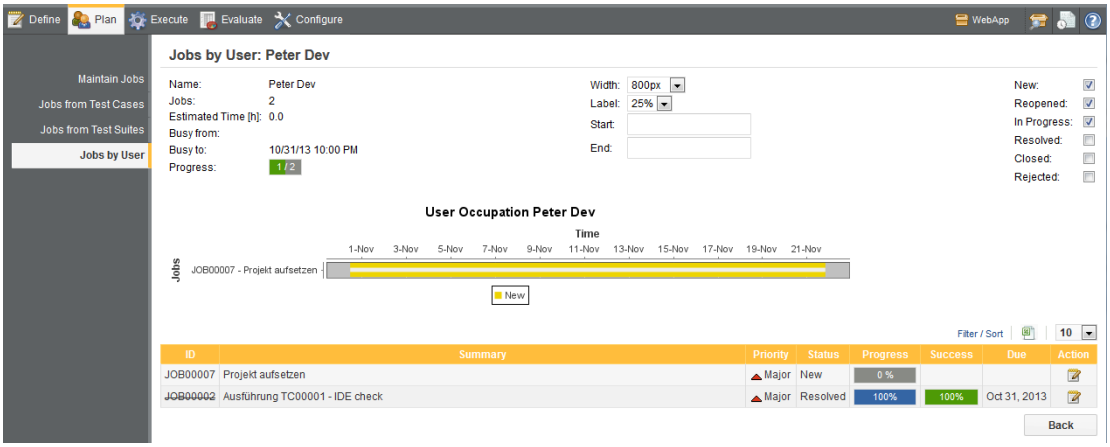


Figure 7.10. The Jobs By User Screen

Chapter 8. Execute

This chapter describes the processes of executing test cases and test suites, continuing interrupted test suite executions and creating issues for tests that failed.

8.1. My Jobs

Feature only available in Klaros-Testmanagement Enterprise Edition

The *My Jobs* page shows the jobs for the active user and allows them to be executed. (Figure 8.1).

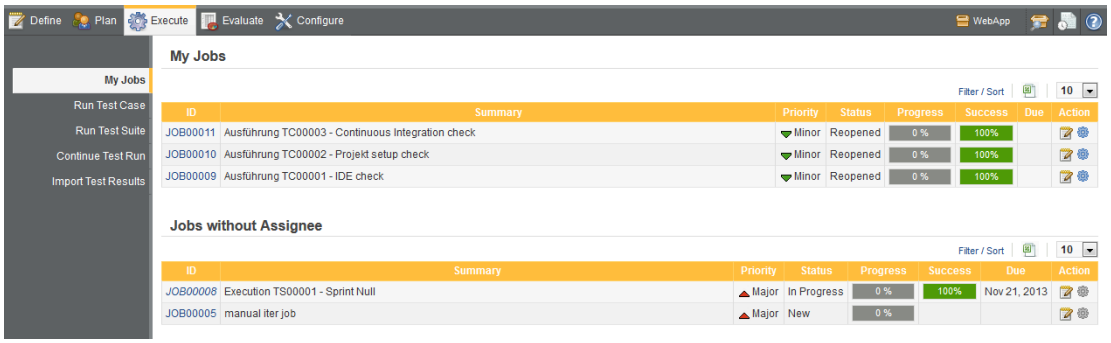


Figure 8.1. The My Jobs Screen

Details such as the number of assigned jobs, the total estimated time for the jobs, the timeframe for the jobs and the number of completed jobs are displayed at the top of this page.

A table displays the details of each individual job. The action column contains a button which allows the user to execute the Job. This process is described in Section 7.1.4, “Executing a Job”.

8.2. Run Test Case

Executing a test case consists of running a test on a specific version of the system under test in a selected test environment.

The *Run a Single Test Case* page shows all test cases in a list. To get a better overview of the available test cases, the sorting and filtering functionality of the *Filter / Sort* panel can be used, as described in Section 5.2.5, “Filtering and Sorting”.

8.2.1. Executing a Single Test Case

Clicking on the action icon of a test case executes that *Test Case* (Figure 8.2).

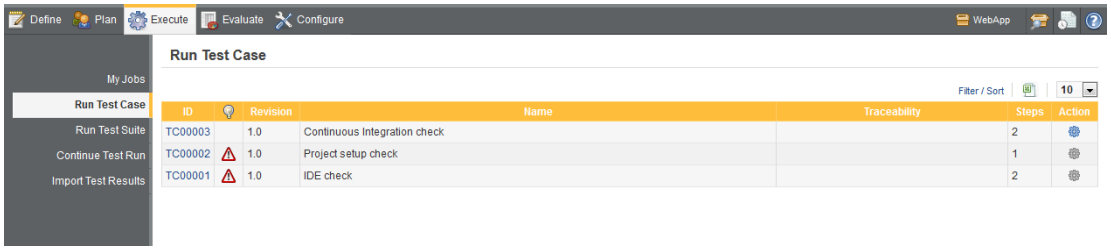


Figure 8.2. The Run Test Case Screen

Warning

Only test cases with the states *Draft*, *Approved* and *Skip* can be executed.

A dialog showing information about the test case will be displayed (Figure 8.3). Clicking the view icon on the right side of the dialog displays more detailed information about the test case.

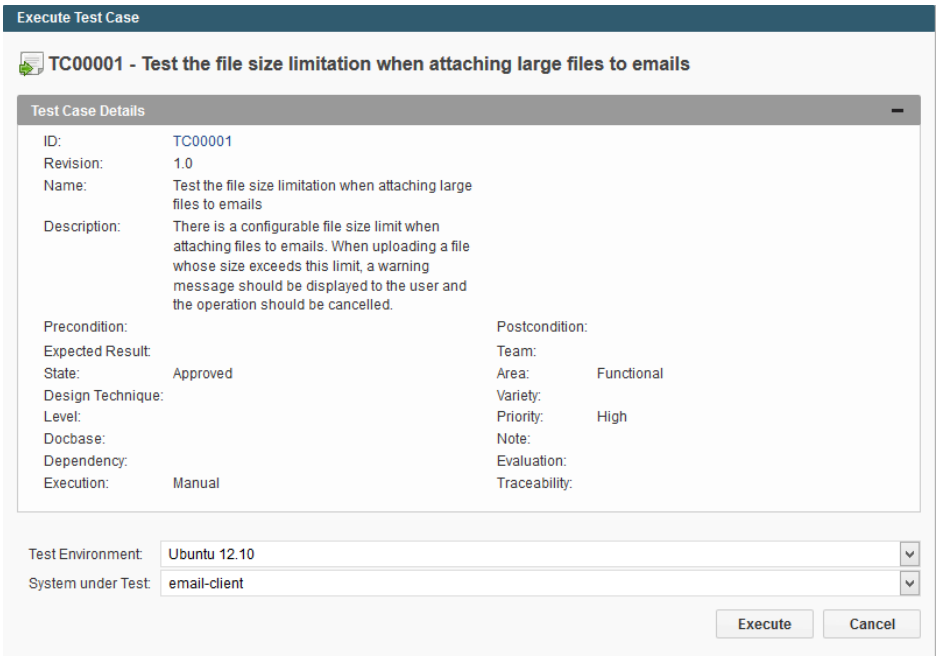


Figure 8.3. The Single Test Case Execution Dialog

Before a test run can be started, the test environment the test is run in and the system under test itself have to be defined. User defined properties may also be set for the test run here.

Creating Test Environments and Systems under Test

For information on how to create a test environment or a system under test, please refer to Chapter 6, *Define*.

Press the **Execute** button to run the test case. By default, the step-by-step instructions screen will open, but this can be changed to the tabular step instructions view by marking the tabular view checkbox.

Note

Pop-ups must be enabled in your browser, or an exception must be added for Klaros-Testmanagement, in order to execute tests manually.

8.2.1.1. The Step-by-step Instructions View

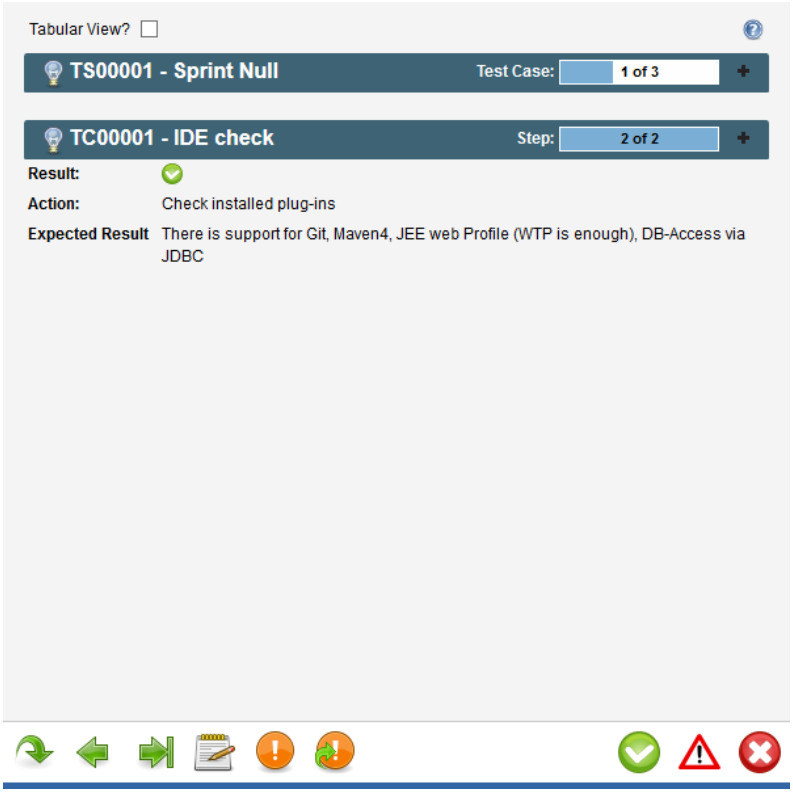


Figure 8.4. The Step-by-step Instructions View

The step by step instructions screen shows the *Action*, *Expected Result*, *Precondition* and *Post-condition* values of the current test step. Depending on the results of the step, the user can click the following buttons:

- **Passed**

  Mark the test step as successfully completed.

- **Error**

  Mark the test step result as *Error*.

- **Failure**

  Mark the test step result as *Failure*.

- **Skip**

  Skip the current test step without changing the result of the test case.

- **Skip All**

Skip all test steps without changing the result of the test case.

- **Create Issue**

  Create an issue related to this test case in an issue management system.

- **Link Issue**

  Link an existing issue from an issue management system to this test case.

- **Back**

  Go back to the last test step to repeat it again or to edit it.

Note

The verdict (e.g. *Passed*, *Skipped* etc.) of already executed test steps is shown in the test runner!

The same process is applied for each test step of the test case.

Clicking the **Next Step** button continues the test even if an error or failure occurred, and clicking the **End Test** button finishes the test execution.

Clicking the **Finish** button, causes Klaros-Testmanagement to show the test run result of the test case (Figure 8.7).

8.2.1.2. The Tabular Step Instructions Screen

Figure 8.5. The Tabular Step Instructions Screen

The tabular step instructions screen shows the *Description*, *Expected Result*, *Precondition*, *Postcondition* and result of all steps, the default result being *Unknown*.

The following actions can be carried out at any time:

- **Skip**

  Skip the current test step without changing the result of the test case.

- **Create Issue**

  Create an issue related to this test case in an issue management system.

- **Finish Run**

  Finish the test run.

- **Passed**

  Mark the test step as successfully completed.

- **Error**

  Mark the test step result as *Error*.

- **Failure**

  Mark the test step result as *Failure*.

Clicking on the result of individual steps allows the user to directly set them.

Clicking the **Finish Run** button, causes Klaros-Testmanagement to show the test run result of the test case (Figure 8.7).

What is the difference between a failure and an error?

A failure is a discrepancy between a computed, observed, or measured value or condition and the true,specified, or theoretically correct value or condition.

An error is the inability of the system to perform the test correctly.

For each error or failure, Klaros-Testmanagement will show the following dialog (Figure 8.6). The **Continue** to the next step while the **Finish** button saves the comment and completes the test case run.

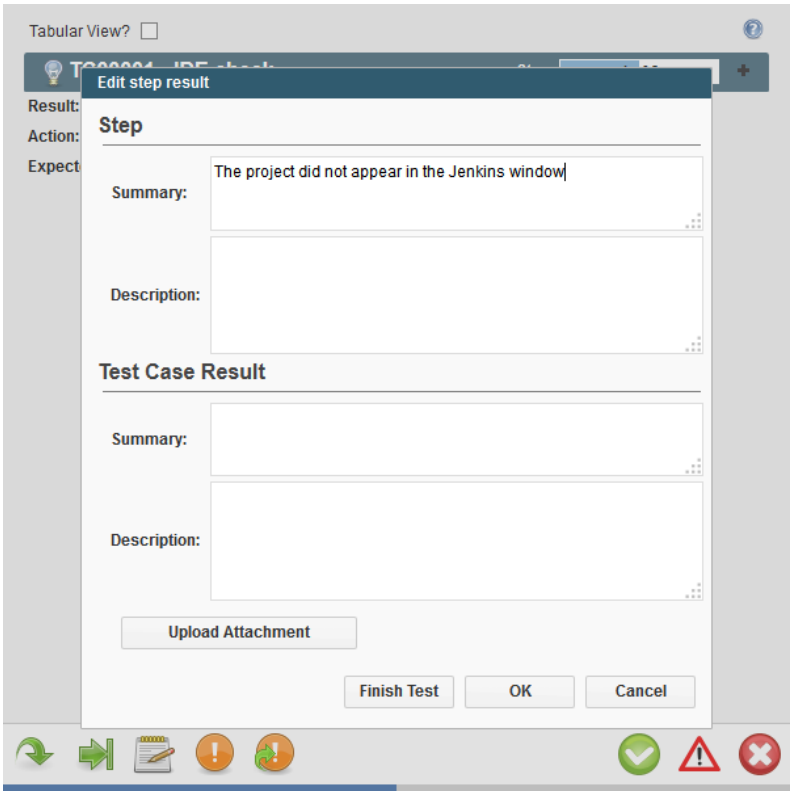


Figure 8.6. The Error or Failure Detected Dialog

Tip

The *Description* and *Summary* fields appear in all comment dialogs.

The test case results screen allows the user to write a description and summary for the test case run as well as for each individual test step in the test run. (Figure 8.7).

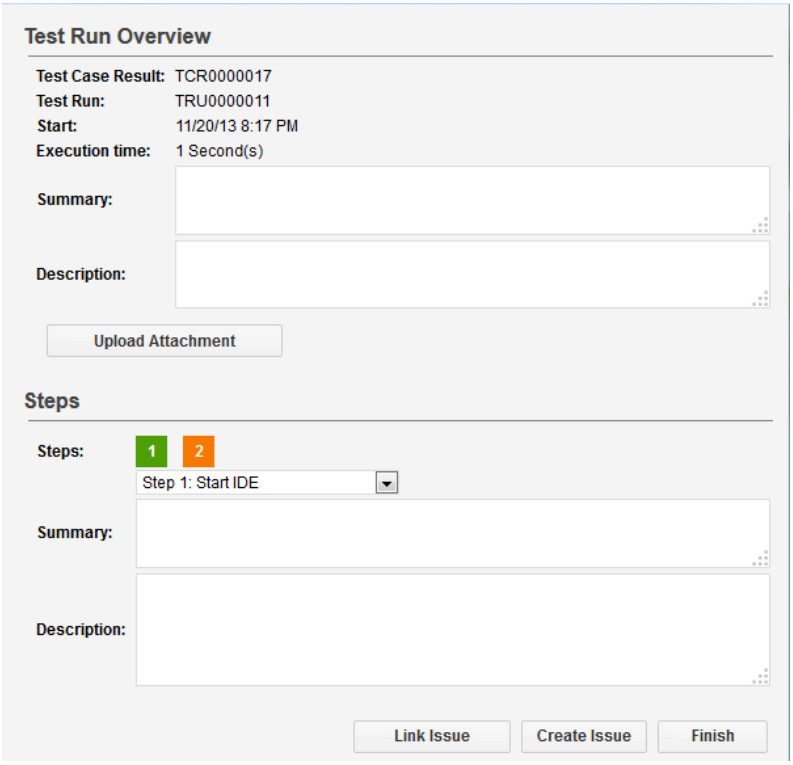


Figure 8.7. The Test Case Results Screen

Note

If there are any issue management systems configured for this project, the **Create Issue** button will be enabled. For details on creating an issue with Klaros-Testmanagement, see Section 8.6, “Creating Issues”.

8.3. Run Test Suite

Executing a Test suite consists of running a defined set of test cases on a specific version of the system under test in a selected test environment.

The *Run Test Suite* page shows all test suites of a project in a list. To get a better overview of the available test suites, the sorting and filtering functionality of the *Filter / Sort* panel can be used as described in Section 5.2.5, “Filtering and Sorting”.

8.3.1. Executing a Test Suite

Clicking on the action icon of a test suite executes that test suite (Figure 8.8).

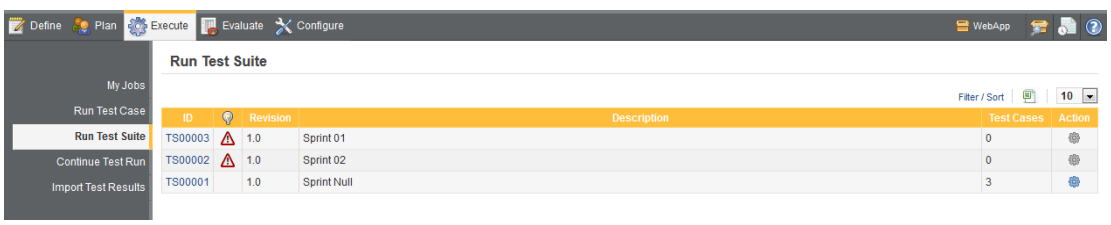


Figure 8.8. The Run Test Suite Screen

Warning

Test cases with the state *Locked* or without any steps will be skipped when executing the test suite.

A dialog showing information about the test suite will be displayed (Figure 8.9). Clicking the icon on the right side of the dialog displays more detailed information about the test suite.

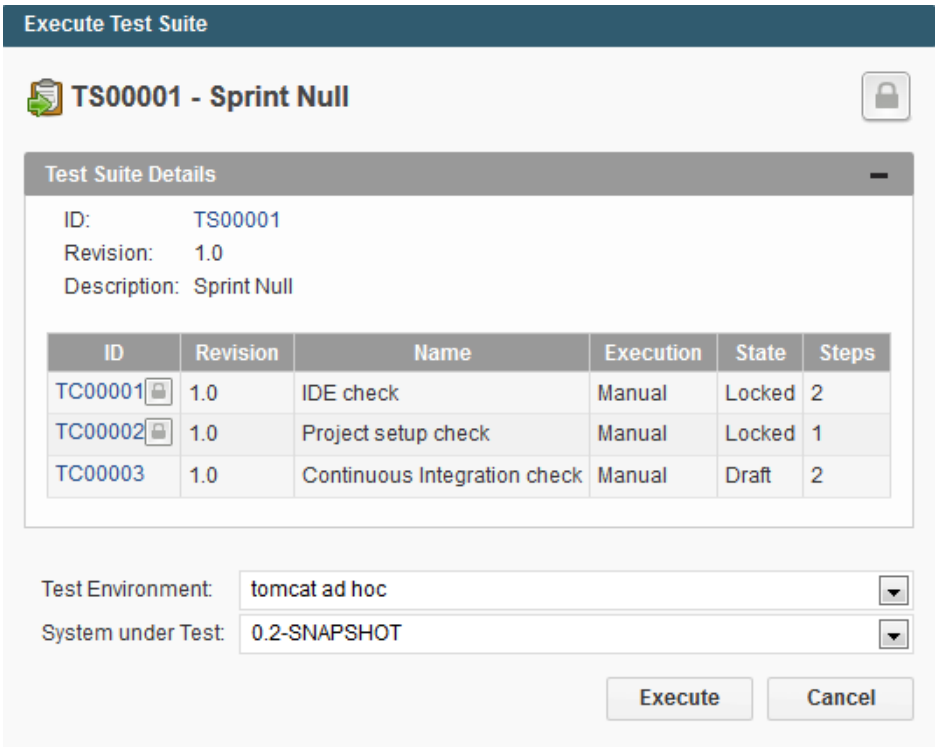

Execute Test Suite
TS00001 - Sprint Null
Test Suite Details
ID: TS00001
Revision: 1.0
Description: Sprint Null
ID Revision Name Execution State Steps
TC00001 1.0 IDE check Manual Locked 2
TC00002 1.0 Project setup check Manual Locked 1
TC00003 1.0 Continuous Integration check Manual Draft 2
Test Environment: tomcat ad hoc
System under Test: 0.2-SNAPSHOT
Execute
Cancel

Figure 8.9. The Detailed Information about the Test Suite Screen

Before a test run is started, a test environment and a system under test must be chosen. user-defined properties may also be set for the test run here.

Press the **Execute** button to run the test suite. Klaros-Testmanagement will show the following pop-up window:

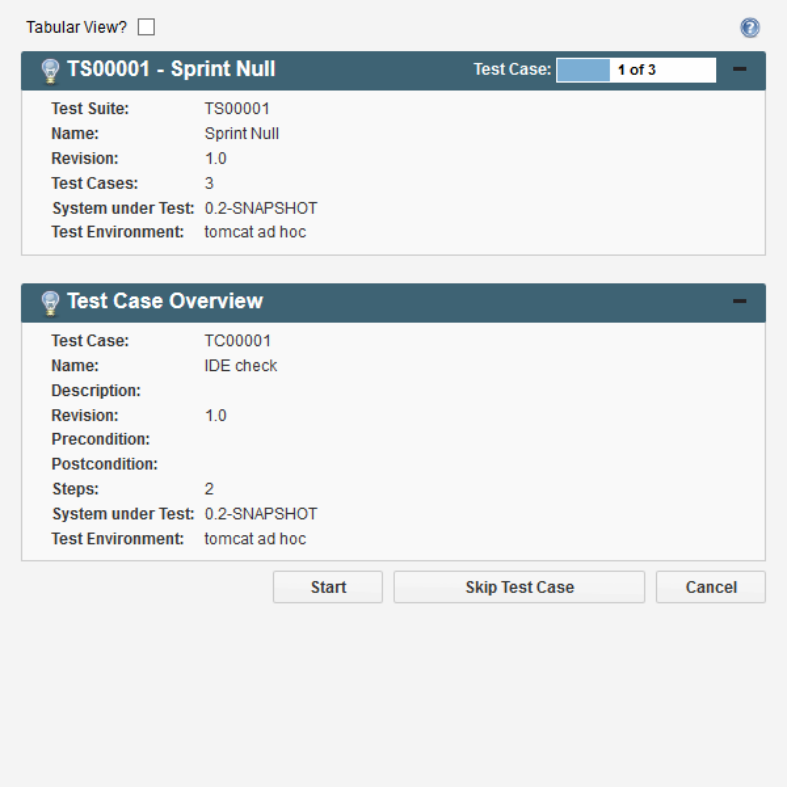


Figure 8.10. The Overview Screen

Note

Pop-ups must be enabled in your browser, or an exception must be added for Klaros-Testmanagement, in order to execute tests manually.

This screen shows the test suite overview and the attachments of the test suite if it has any. Clicking the icon starts the download of the corresponding attachment.

Click the **Start** button to begin the manual test run. By default, the step-by-step instructions screen will open, but this can be changed to the tabular step instructions view by marking the tabular view checkbox.

The test suite results screen allows the user to write a description and summary for the test suite run as well as for each individual test case and test step in the test run. (Figure 8.11).

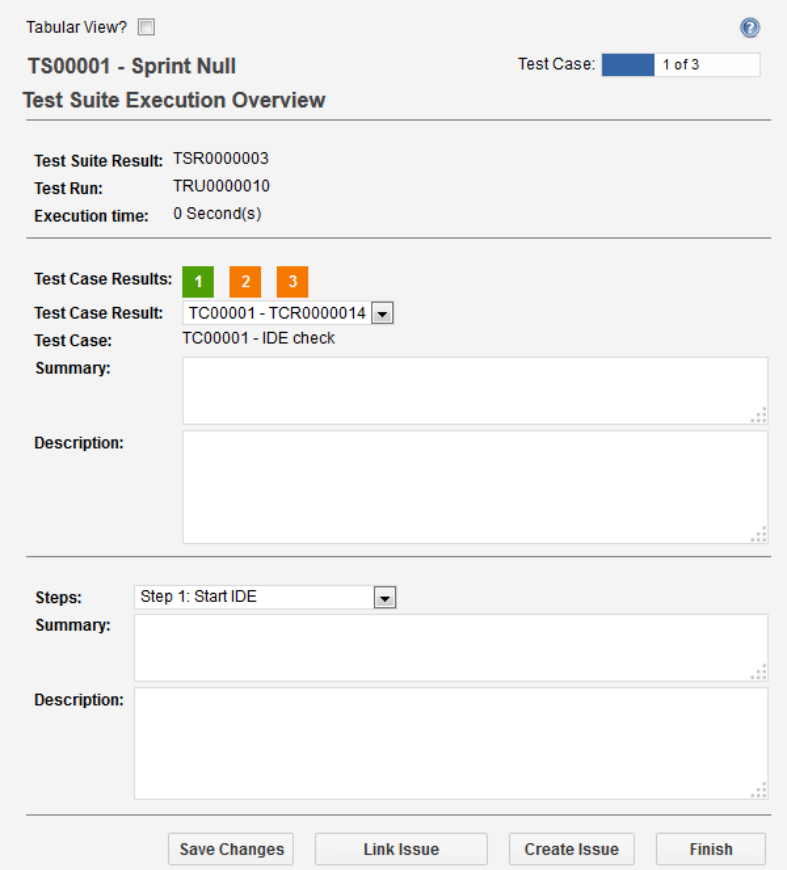


Figure 8.11. The Test Suite Results Screen

Note

If there are any issue management systems configured for this project, the **Create Issue** button will be enabled. For instructions on creating an issue with Klaros-Testmanagement see Section 8.6, “Creating Issues”.

8.4. Continue Test Run

Sometimes a test run has to be suspended due to working hours or external conditions which prevent further test case executions. A tester may stop executing a test run at any time without losing any previously entered test data.

A test run which is interrupted in its execution can be continued later. The execution resumes with the first test case step with no result.

The *Continue Test Run* page lists all test runs in the selected project which have not been finished. To get a better overview of the available interrupted test runs, the sorting and filtering functionality of the *Filter / Sort* panel can be used, as described in Section 5.2.5, “Filtering and Sorting”.

Note

The *Miscellaneous* settings page contains a setting which prevents users with the role *Tester* from seeing interrupted test runs of jobs from other users (Section 10.5.1, “Miscellaneous Settings”).

8.4.1. Continuing a Suspended Test Run

To continue the execution of a stopped/suspended test run click on the icon of the test run (Figure 8.12). To delete a suspended test run click on the icon in the action column.

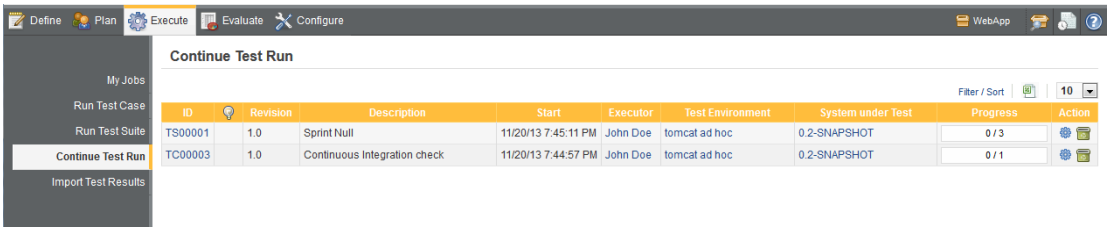


Figure 8.12. The Continue Test Run Screen

Section 8.2, “Run Test Case” describes the process of executing a test suite. The new test results are merged automatically with the suspended test suite run results.

8.5. Import Test Results

Klaros-Testmanagement allows users to import test results from several test frameworks. If the related test cases are not already defined in the Klaros-Testmanagement database, they will be automatically created. The following frameworks are supported:

AUnit	*AUnit* provides mechanisms by which developers can test their AspectJ aspects in isolation in support of Test Driven Development.
Boost Test	The *Boost Test* library provides a matched set of components for writing test programs, organizing tests into simple test cases and test suites, and controlling their runtime execution.
Check	*Check* is a unit testing framework for C.
CppTest	*CppTest* is a portable and powerful, yet simple, unit testing framework for handling automated tests in C++. The focus lies on usability and extendability.
CppUnit	*CppUnit* is a C++ unit testing framework.
ctest	*ctest* is the testing driver provided by CMake.
CUnit	*CUnit* is a lightweight system for writing, administering, and running unit tests in C.
Free Pascal Unit	*Free Pascal Unit* is a port to Free Pascal of the JUnit core framework.

GUIdancer	*GUIdancer* is an Eclipse-based tool for automated functional testing through the Graphical User Interface (GUI).
Jubula	*Jubula* provides automated functional GUI testing for various types of applications.
JUnit	*JUnit* is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.
MbUnit	*MbUnit* is an extensible unit testing framework for the .NET Framework that takes in and goes beyond xUnit pattern testing. MbUnit is part of the Gallio bundle.
MSTest	*MSTest* is the command-line command that is used to run tests. This command has several options you can use to customize your test run.
NUnit	*NUnit* is a unit-testing framework for all .Net languages.It is written entirely in C# and has been completely redesigned to take advantage of many .NET language features, for example custom attributes and other reflection related capabilities.
PHPUnit	*PHPUnit* is a member of the xUnit family of testing frameworks and provides both a framework that makes the writing of tests easy as well as the functionality to easily run the tests and analyse their results.
QF-Test	*QF-Test* is a professional tool for automated testing of Java and Web applications with a graphical user interface.
QTestLib	The *QTestLib* framework is a tool for unit testing Qt based applications and libraries.
Ranorex	*Ranorex* is a GUI test automation framework for testing of desktop, web-based and mobile applications.
Selenium	Selenium is a web browser automation tool primarily used for automated testing of web apps. Selenium is able to produce *JUnit*-compatible test results, which can be imported into Klaros-Testmanagement.
Test Complete	*TestComplete* is a cross platform automated GUI testing tool from SmartBear Software.
TestNG	*TestNG* is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use. TestNG is able to produce *JUnit*-compatible test results, which can be imported into Klaros-Testmanagement.
TUSAR	*TUSAR* stands for Thales Unified Sofware Analysis Report. It is a generic metric format composed of 4 categories: Coverage, Measure, Test and Violations.

UnitTest++	*UnitTest++* is a lightweight unit testing framework for C++. It was for test-driven development on a wide variety of platforms. Simplicity, portability, speed, and a small footprint are all very important aspects.
Valgrind	*Valgrind* is an instrumentation framework for building dynamic analysis tools.
xUnit.net	*xUnit.net* is a free, open source, community-focused unit testing tool for the .NET Framework. xUnit.net works with ReSharper, CodeRush, TestDriven.NET and Xamarin.

Support for Frameworks not listed here

Even if you did not find your favorite testing framework here this does not mean that it is not already supported.

Many tools offer to export their result files in the JUnit XML format (e.g. Selenium, Robot and others) which is a supported format.

If you like to see an additional format supported, please contact support@verit.de.

To start the import, an XML file with results from a supported test framework must be uploaded. When the file upload is completed it is mandatory to select the system under test and the test environment these results should be assigned to. The format of the result file is typically detect automatically.

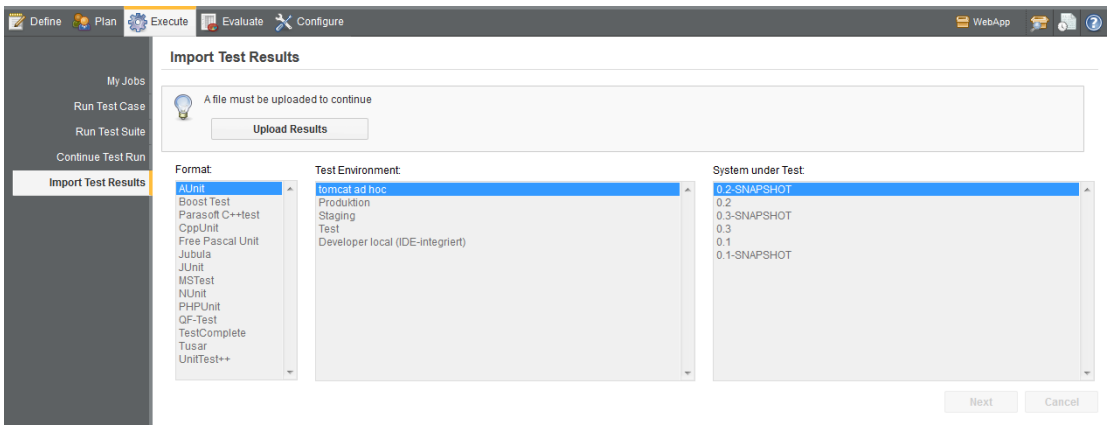


Figure 8.13. Import Results Page

Clicking the **Next** button opens the *Assign Test Results to Test Cases* page. The test case to assign each test result to can be selected using the corresponding drop-down box. Each result must be assigned to a different test case. If no test case is selected here a new test case is automatically created upon import. When the option *Create additional test suite result* is selected test suite results are automatically created for the corresponding test suite information contained in the result file. This information may vary with the import format. Additionally a corresponding test suite is created for the test suite result if it did not yet exist.

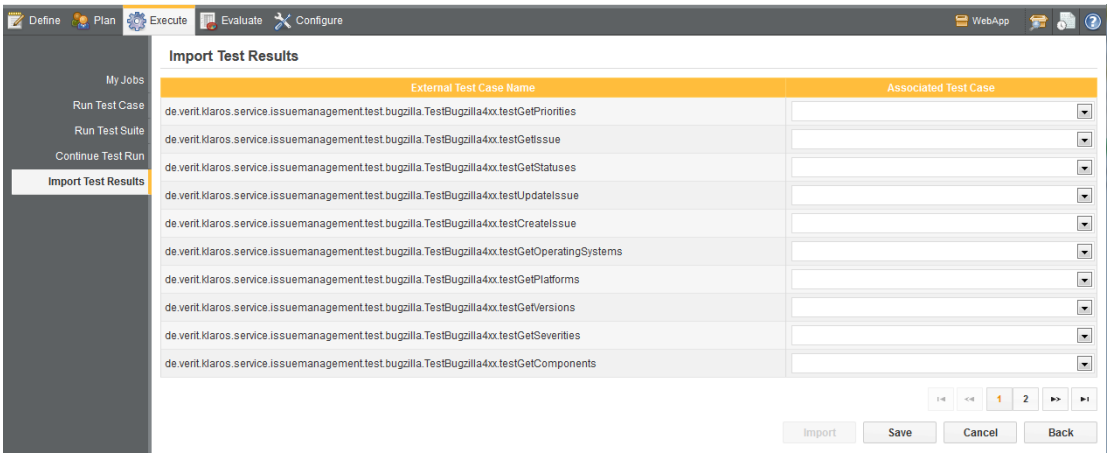


Figure 8.14. The Import Test Results Page

Pressing the **Import** button will import all results into Klaros-Testmanagement.

8.6. Creating Issues

After the execution of a test case or test suite has been finished, an issue in an issue management system can be created. In order to do this, at least one issue management system has to be configured and also available in the selected project. The configuration of an issue management system is described in Section 10.6.1, "Issue Management", and Section 6.1.6.1, "Editing the Issue Management Systems in a Project" contains instructions on how to add an issue management system to a project.

Clicking on the **Create Issue** button on the results page which is displayed after the execution of a test case or test suite opens the *Create Issue* page.

By default, the system uses the credentials entered at the login screen to authenticate the user against the issue management system. If the credentials entered at login do not work, and the issue management system has not yet been used during the session, Klaros-Testmanagement will request new authentication information in a dialog.

The content of the *Issue Management* page is mostly dependent on the issue management system in use and its capabilities, but all of the pages for the various issue management systems have some elements in common:

Issue Management System
: When there are more than one issue management systems configured for the project, the issue management system where the new issue will be created can be selected in the issue management system drop-down field.

Save
: The **Save** button creates or updates an issue in the selected issue management system.

Cancel
: The **Cancel** button discards unsaved changes and directs the user back to the test execution.

New
: After an issue has been saved, an additional **New** button is provided to allow the creation of more than one issue (e.g. in another issue management system).

The following fields are available for the JIRA issue management system:

Summary	A short summary of the issue.
Issue Type	The type of the issue, e.g. a bug or an improvement.
Priority	The priority of the issue.
Components	The components of the system that are affected, multiple selections are allowed here.
Affected Versions	The version of the system in which the issue appeared, multiple selections are allowed here.
Fixed Versions	The version of the system in which the issue has been fixed. Multiple selections are allowed here.
Due Date	The date when the issue should be fixed.
Description	A longer description of the issue.
Test Environment	A description of the test environment in which the issue occurred.

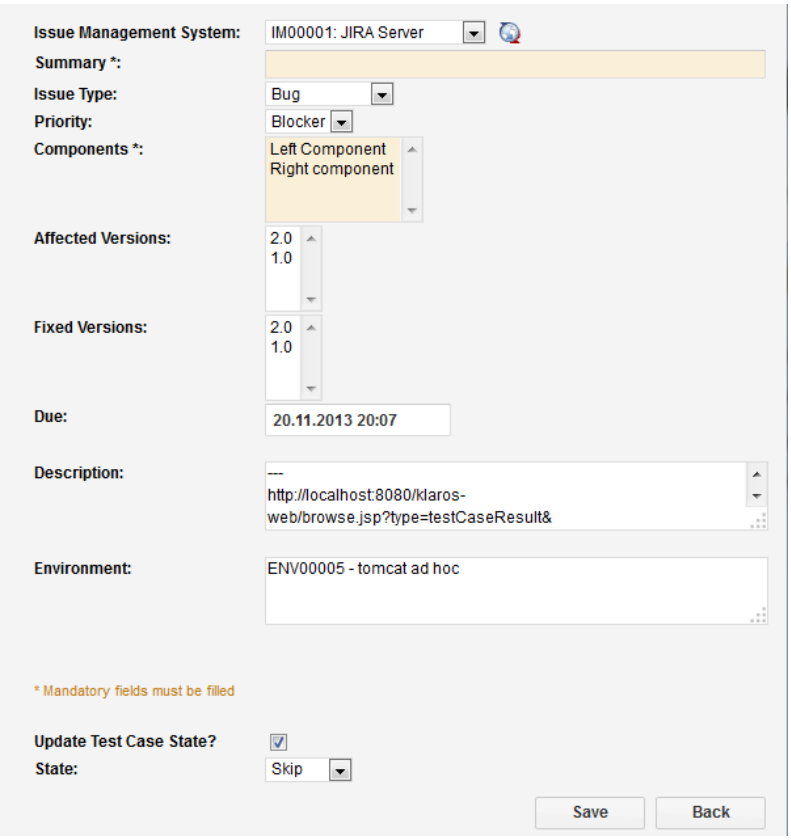


Figure 8.15. Jira Issue Page

The following fields are available for the Trac issue management system:

Summary	A short summary of the issue.
Issue Type	The type of the issue, e.g. a defect or enhancement.
Importance	The priority of the issue.

Components	The component of the system that is affected.
Version	The version of the system that is affected.
Milestone	The milestone of the system that is affected.
Description	A detailed description of the issue.
Keywords	Keywords for the issue.

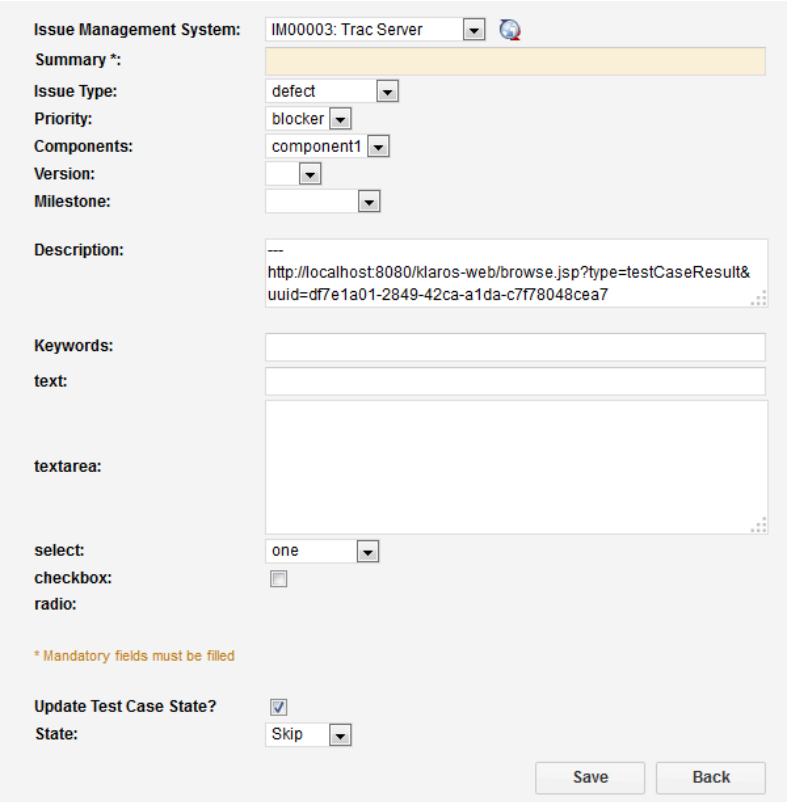


Figure 8.16. Trac Issue Page

The following fields are available for the `Redmine` issue management system:

Summary	A short summary of the issue.
Issue Type	The type of the issue, e.g. a defect or enhancement.
Importance	The priority of the issue.
Assignee	The assignee of the issue.
Version	The version of the system that is affected.
Category	The category of the system that is affected.
Description	A detailed description of the Issue.
Estimated time [hours]	The estimated time in hours for completing the issue.

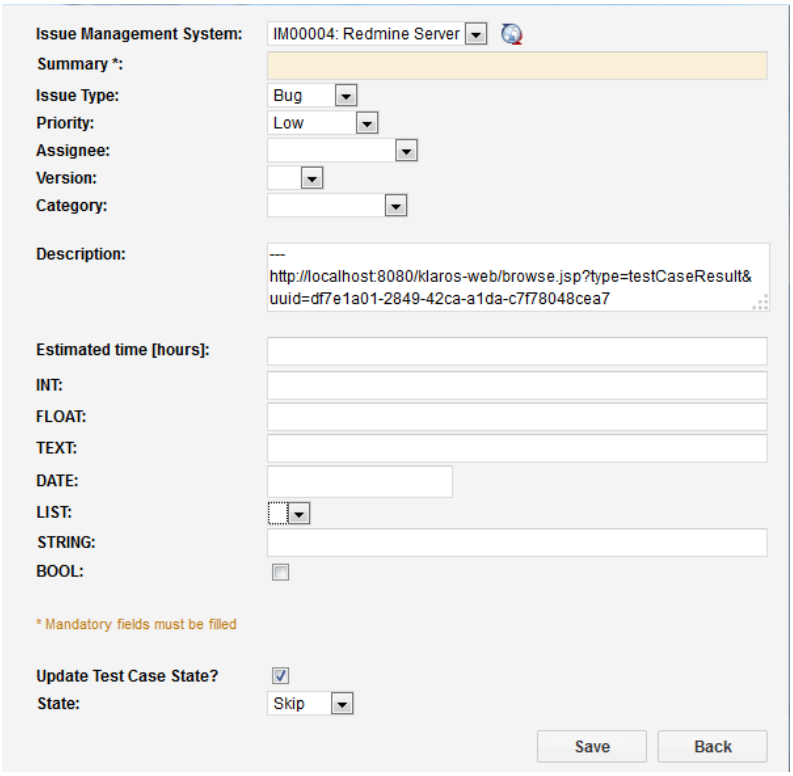


Figure 8.17. Redmine Issue Page

The following fields are available for the `Bugzilla` issue management system:

Summary — A short summary of the issue.

Platform — The platform the issue appears on.

Importance — The priority and type of the issue.

Components — The components of the issue.

Version — The version of the system that is affected.

Milestone — The milestone of the system that is affected.

Description — A detailed description of the issue.

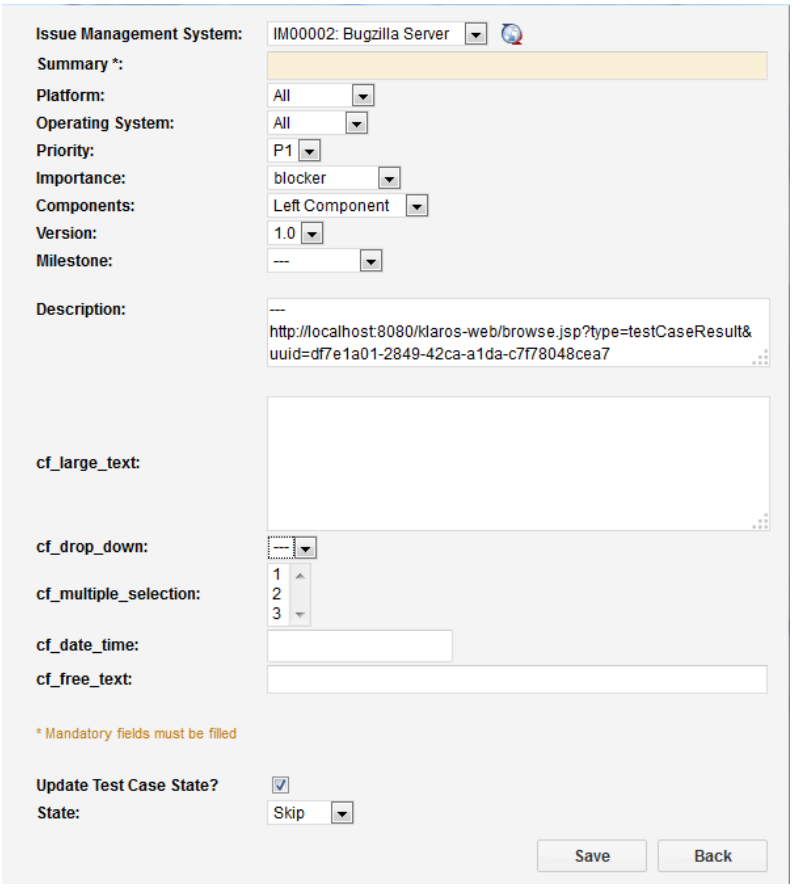


Figure 8.18. Bugzilla Issue Page

The following fields are available for the `Mantis` issue management system:

Summary	A short summary of the issue.
Reproducibility	The ease of reproduction of the issue.
Importance	The priority and type of the issue.
Version	The version of the system that is affected.
Category	The category of the issue.
Status	The current status of the issue.
Resolution	The current resolution of the issue.
Description	The detailed description of the issue.
Additional Information	A text field for additional information about the issue.

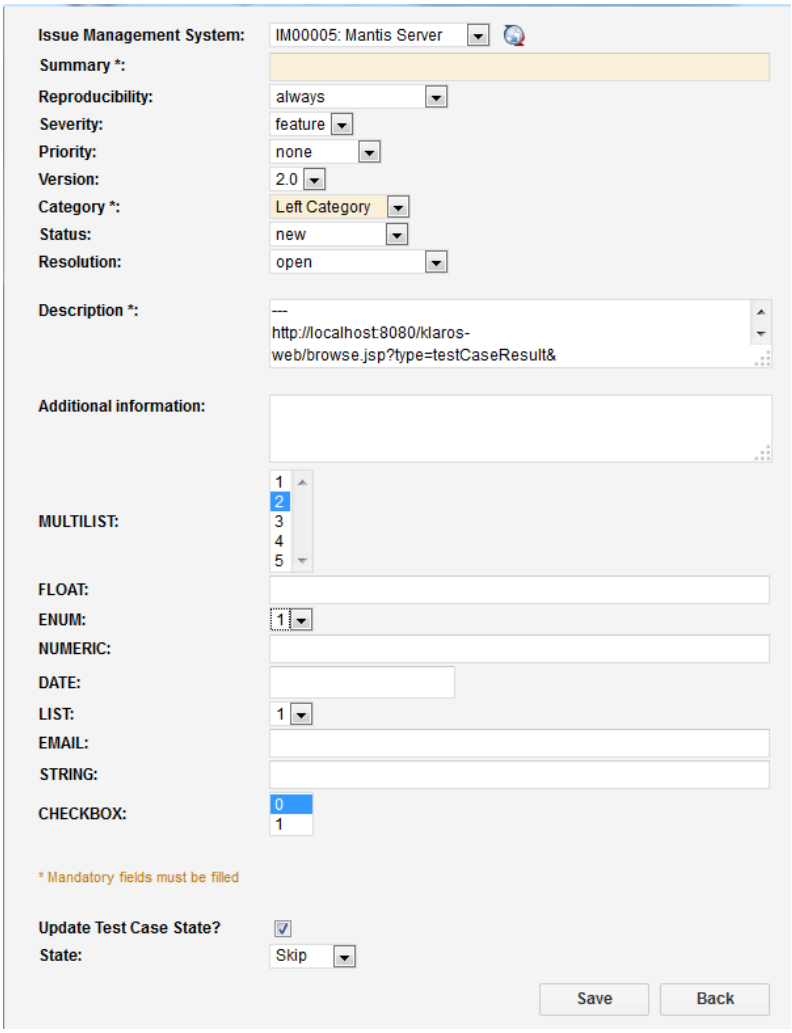


Figure 8.19. Mantis Issue Page

Tip

To change the state of the related test case just activate the **Update Test Case State?** button and choose the new state from the drop down box.

For more detailed information about each field, please consult the documentation of the respective issue management system.

The created issues can be inspected in the *Issues by Test Case* page of the *Evaluate* section see Section 9.7.1, “Issues by Test Case” for more details.

8.7. Creating Review Jobs

Feature only available in Klaros-Testmanagement Enterprise Edition

If a tester notices a flaw in a test case or test suite description at any point during test execution, he or she can schedule a review job by pressing the building-site helmet icon ( ). This opens a dialog window in which the details of the review job can be entered:

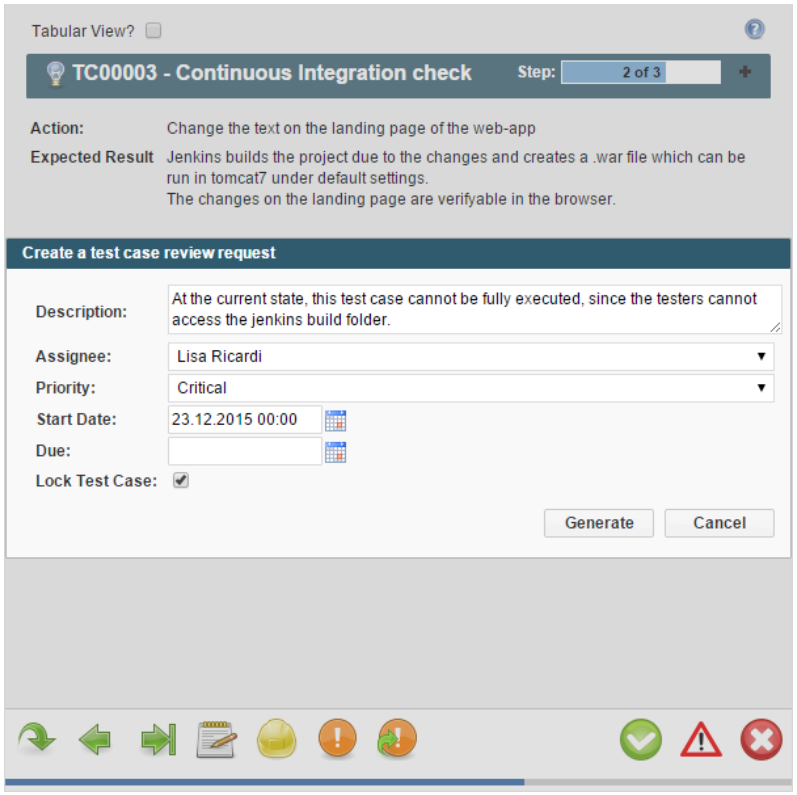

Tabular View?
TC00003 - Continuous Integration check
Step: 2 of 3
Action:
Change the text on the landing page of the web-app
Expected Result
Jenkins builds the project due to the changes and creates a .war file which can be run in tomcat7 under default settings.
The changes on the landing page are verifyable in the browser.
Create a test case review request
Description:
At the current state, this test case cannot be fully executed, since the testers cannot access the jenkins build folder.
Assignee:
Lisa Ricardi
Priority:
Critical
Start Date:
23.12.2015 00:00
Due:
Lock Test Case:
Generate
Cancel

Figure 8.20. Creating a Review Job during Execution

Chapter 9. Evaluate

This chapter describes how Klaros-Testmanagement can be used to gather information about the selected project and evaluate its data.

9.1. Dashboard

The dashboard shows basic statistics about the active project and provides the links to the overview reports. An example of a dashboard is shown in Figure 9.1.

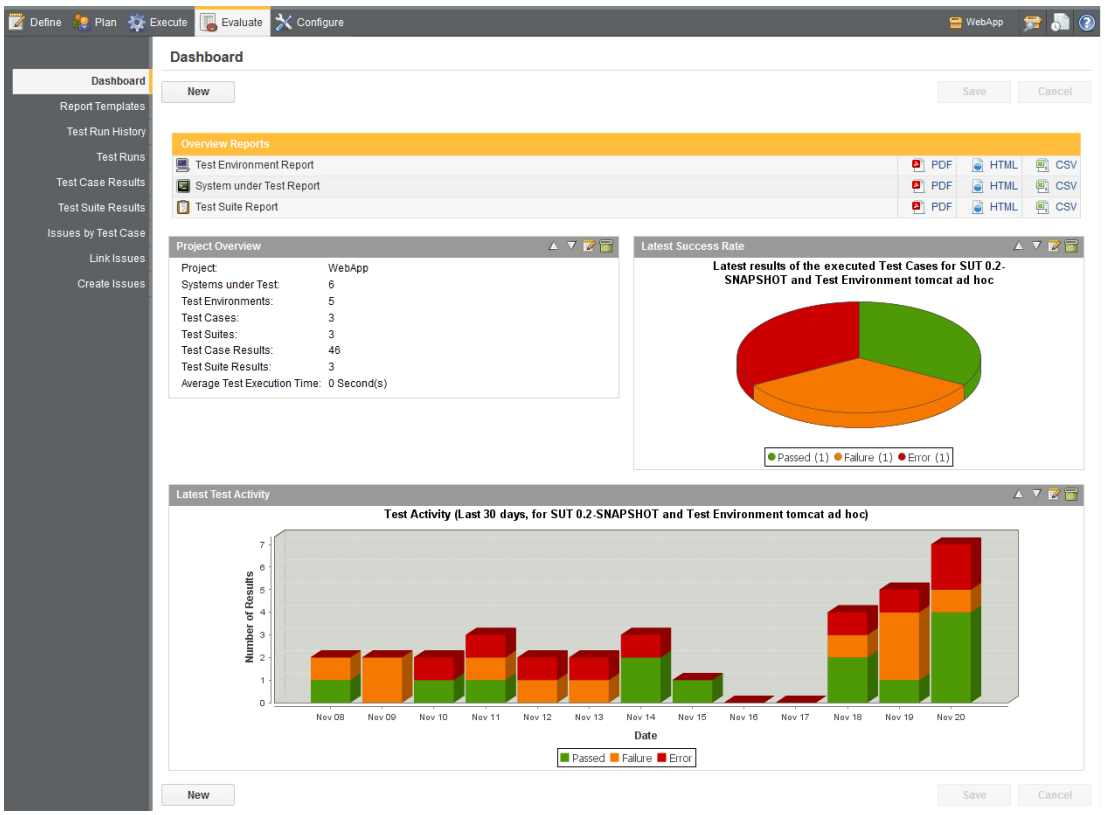


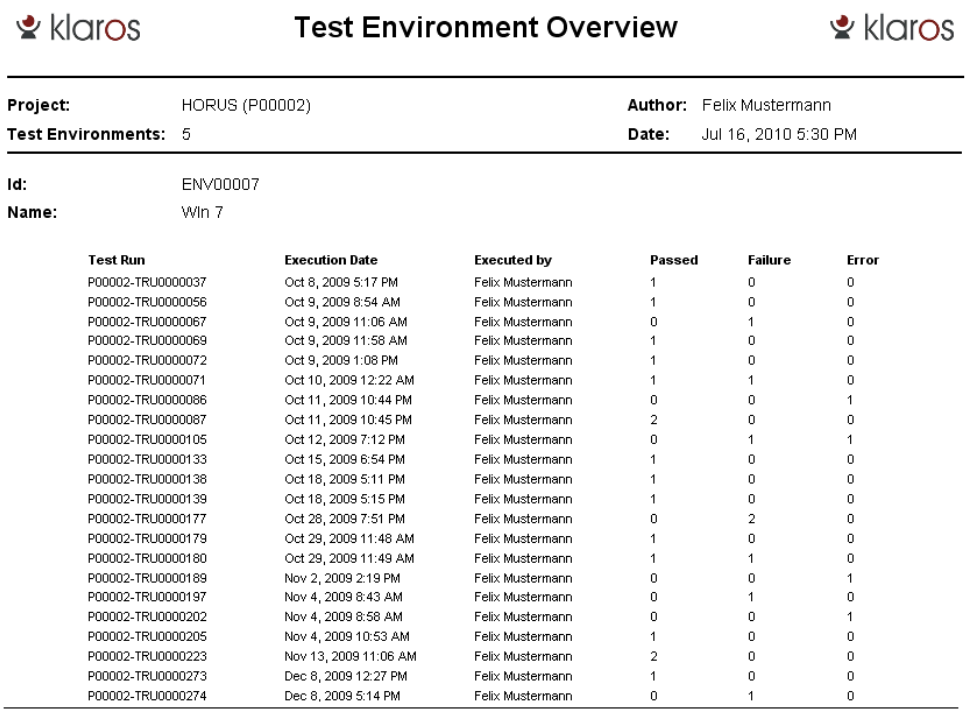
Figure 9.1. The Dashboard Screen

9.1.1. The Overview Reports

The upper part of the dashboard shows a table with the available overview reports. The following three types of reports are available:

- *Test Environment Overview*

  The *Test Environment Overview* report shows the test runs that have been executed in each test environment containing the execution date and quantity of success, failure and error results. An example of the report is shown in Figure 9.2.



klaros

Test Environment Overview

klaros

Project:	HORUS (P00002)	**Author:**	Felix Mustermann
Test Environments:	5	**Date:**	Jul 16, 2010 5:30 PM

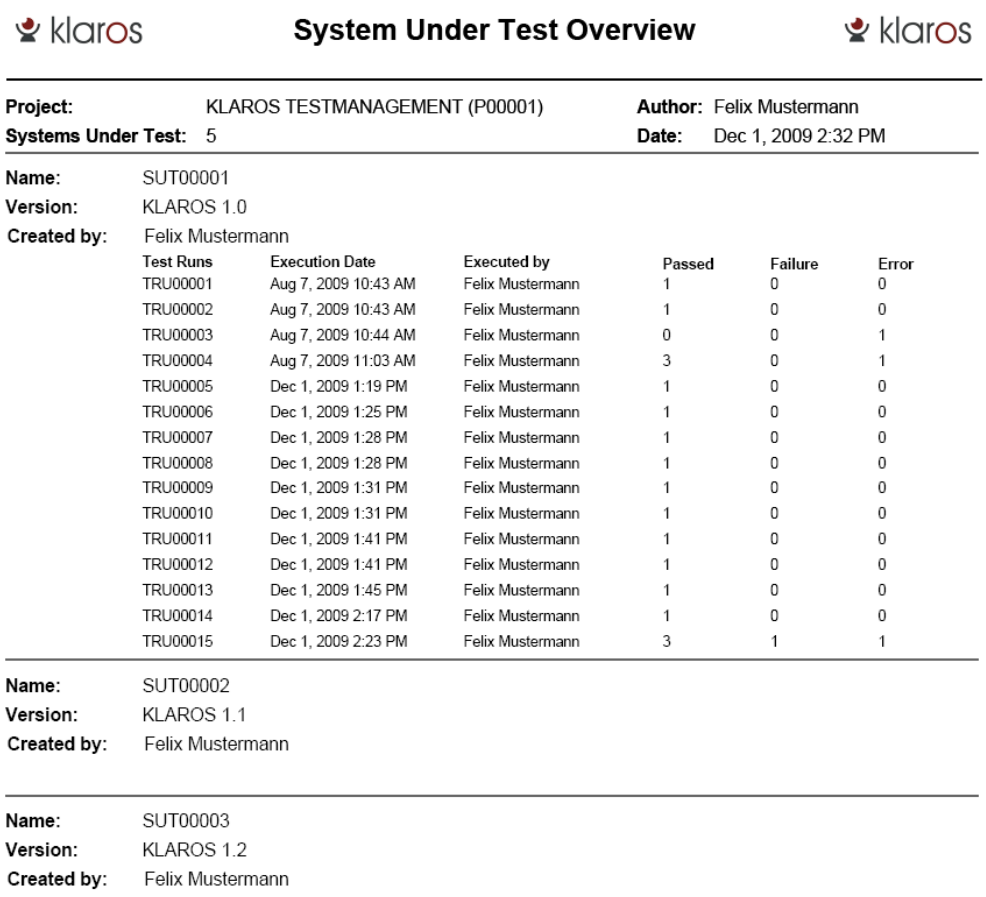
Id: ENV00007
Name: Win 7

Test Run	Execution Date	Executed by	Passed	Failure	Error
P00002-TRU0000037	Oct 8, 2009 5:17 PM	Felix Mustermann	1	0	0
P00002-TRU0000056	Oct 9, 2009 8:54 AM	Felix Mustermann	1	0	0
P00002-TRU0000067	Oct 9, 2009 11:06 AM	Felix Mustermann	0	1	0
P00002-TRU0000069	Oct 9, 2009 11:58 AM	Felix Mustermann	1	0	0
P00002-TRU0000072	Oct 9, 2009 1:08 PM	Felix Mustermann	1	0	0
P00002-TRU0000071	Oct 10, 2009 12:22 AM	Felix Mustermann	1	1	0
P00002-TRU0000086	Oct 11, 2009 10:44 PM	Felix Mustermann	0	0	1
P00002-TRU0000087	Oct 11, 2009 10:45 PM	Felix Mustermann	2	0	0
P00002-TRU0000105	Oct 12, 2009 7:12 PM	Felix Mustermann	0	1	1
P00002-TRU0000133	Oct 15, 2009 6:54 PM	Felix Mustermann	1	0	0
P00002-TRU0000138	Oct 18, 2009 5:11 PM	Felix Mustermann	1	0	0
P00002-TRU0000139	Oct 18, 2009 5:15 PM	Felix Mustermann	1	0	0
P00002-TRU0000177	Oct 28, 2009 7:51 PM	Felix Mustermann	0	2	0
P00002-TRU0000179	Oct 29, 2009 11:48 AM	Felix Mustermann	1	0	0
P00002-TRU0000180	Oct 29, 2009 11:49 AM	Felix Mustermann	1	1	0
P00002-TRU0000189	Nov 2, 2009 2:19 PM	Felix Mustermann	0	0	1
P00002-TRU0000197	Nov 4, 2009 8:43 AM	Felix Mustermann	0	1	0
P00002-TRU0000202	Nov 4, 2009 8:58 AM	Felix Mustermann	0	0	1
P00002-TRU0000205	Nov 4, 2009 10:53 AM	Felix Mustermann	1	0	0
P00002-TRU0000223	Nov 13, 2009 11:06 AM	Felix Mustermann	2	0	0
P00002-TRU0000273	Dec 8, 2009 12:27 PM	Felix Mustermann	1	0	0
P00002-TRU0000274	Dec 8, 2009 5:14 PM	Felix Mustermann	0	1	0

Figure 9.2. The Test Environment Overview Report Layout

- *SUT Overview*

  The *SUT Overview* report shows the test runs that have been executed for a system under test containing the execution date and quantity of success, failure and error results. An example of the report is shown in Figure 9.3.



klaros **System Under Test Overview** klaros

Project: KLAROS TESTMANAGEMENT (P00001) **Author:** Felix Mustermann
Systems Under Test: 5 **Date:** Dec 1, 2009 2:32 PM

Name: SUT00001
Version: KLAROS 1.0
Created by: Felix Mustermann

Test Runs	Execution Date	Executed by	Passed	Failure	Error
TRU00001	Aug 7, 2009 10:43 AM	Felix Mustermann	1	0	0
TRU00002	Aug 7, 2009 10:43 AM	Felix Mustermann	1	0	0
TRU00003	Aug 7, 2009 10:44 AM	Felix Mustermann	0	0	1
TRU00004	Aug 7, 2009 11:03 AM	Felix Mustermann	3	0	1
TRU00005	Dec 1, 2009 1:19 PM	Felix Mustermann	1	0	0
TRU00006	Dec 1, 2009 1:25 PM	Felix Mustermann	1	0	0
TRU00007	Dec 1, 2009 1:28 PM	Felix Mustermann	1	0	0
TRU00008	Dec 1, 2009 1:28 PM	Felix Mustermann	1	0	0
TRU00009	Dec 1, 2009 1:31 PM	Felix Mustermann	1	0	0
TRU00010	Dec 1, 2009 1:31 PM	Felix Mustermann	1	0	0
TRU00011	Dec 1, 2009 1:41 PM	Felix Mustermann	1	0	0
TRU00012	Dec 1, 2009 1:41 PM	Felix Mustermann	1	0	0
TRU00013	Dec 1, 2009 1:45 PM	Felix Mustermann	1	0	0
TRU00014	Dec 1, 2009 2:17 PM	Felix Mustermann	1	0	0
TRU00015	Dec 1, 2009 2:23 PM	Felix Mustermann	3	1	1

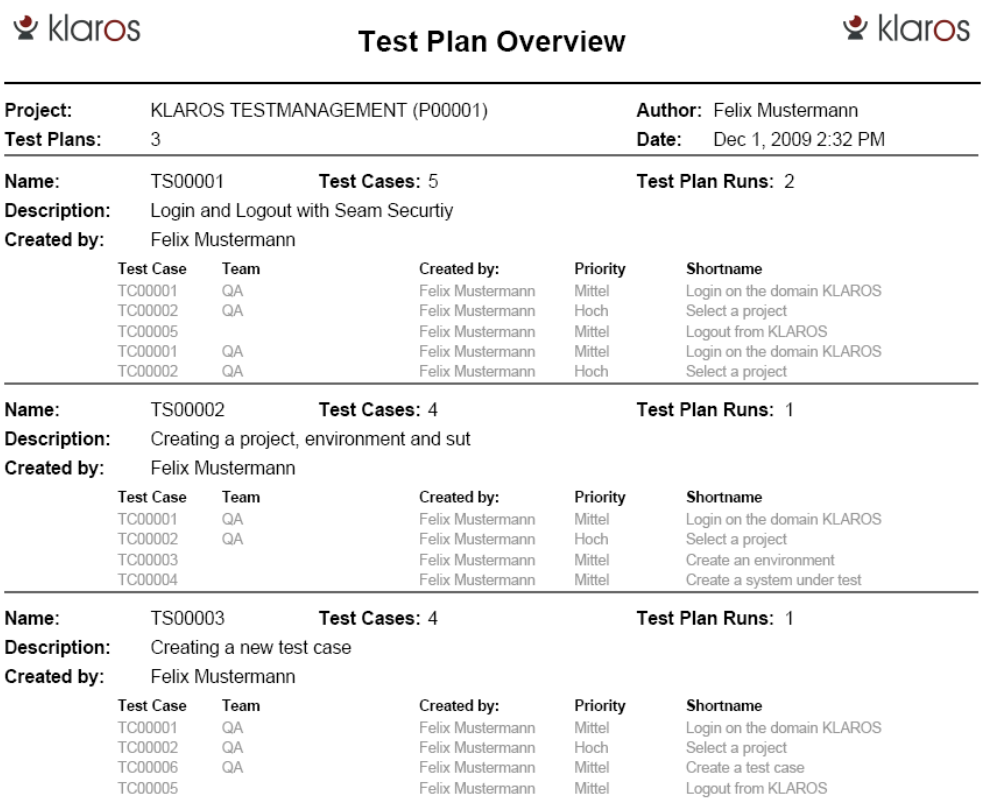
Name: SUT00002
Version: KLAROS 1.1
Created by: Felix Mustermann

Name: SUT00003
Version: KLAROS 1.2
Created by: Felix Mustermann

Figure 9.3. The SUT Overview Report Layout

- *Test Suite Overview*

  The *Test Suite Overview* report shows the selected test suite, containing its test cases, description, how many times it was executed and with what results. An example report is shown in Figure 9.4.



klaros

Test Plan Overview

klaros

Project: KLAROS TESTMANAGEMENT (P00001) **Author:** Felix Mustermann
Test Plans: 3 **Date:** Dec 1, 2009 2:32 PM

Name: TS00001 **Test Cases:** 5 **Test Plan Runs:** 2
Description: Login and Logout with Seam Securtiy
Created by: Felix Mustermann

Test Case	Team	Created by:	Priority	Shortname
TC00001	QA	Felix Mustermann	Mittel	Login on the domain KLAROS
TC00002	QA	Felix Mustermann	Hoch	Select a project
TC00005		Felix Mustermann	Mittel	Logout from KLAROS
TC00001	QA	Felix Mustermann	Mittel	Login on the domain KLAROS
TC00002	QA	Felix Mustermann	Hoch	Select a project

Name: TS00002 **Test Cases:** 4 **Test Plan Runs:** 1
Description: Creating a project, environment and sut
Created by: Felix Mustermann

Test Case	Team	Created by:	Priority	Shortname
TC00001	QA	Felix Mustermann	Mittel	Login on the domain KLAROS
TC00002	QA	Felix Mustermann	Hoch	Select a project
TC00003		Felix Mustermann	Mittel	Create an environment
TC00004		Felix Mustermann	Mittel	Create a system under test

Name: TS00003 **Test Cases:** 4 **Test Plan Runs:** 1
Description: Creating a new test case
Created by: Felix Mustermann

Test Case	Team	Created by:	Priority	Shortname
TC00001	QA	Felix Mustermann	Mittel	Login on the domain KLAROS
TC00002	QA	Felix Mustermann	Hoch	Select a project
TC00006	QA	Felix Mustermann	Mittel	Create a test case
TC00005		Felix Mustermann	Mittel	Logout from KLAROS

Figure 9.4. The Test Suite Overview Report Layout

The reports can be generated in three different file formats:

- PDF
- HTML
- CSV

Clicking on a file type icon generates the report will in the selected format.

9.1.2. The Dashboard Reports

The dashboard is a container for multiple reports. Each report has the same structure: It consists of a title bar and an area in which the report is displayed. The title bar of the report contains the reports name, and an edit button ( ). When the edit button is clicked, the parameters of the report can be configured in the resulting dialog. The changes in the configuration can be applied to the report by clicking the **Apply** button or discarded by clicking the **Cancel** button. Additionally the name of the report can be configured by clicking on the name in the title bar of the report.

The configured dashboard can be persisted by clicking the **Save** button in the upper right corner of the dashboard. If the **Cancel** button is pressed, the changes in the configuration of the dashboard are discarded.

All reports can show data from the currently selected project or from a project selected in the report settings.

Klaros-Testmanagement Community Edition comes with three predefined reports:

- The *Project Overview Report*
- The *Latest Success Rate Report*
- The *Test Activity Report*

These reports are described in the following sections.

In addition, Klaros-Testmanagement Enterprise Edition contains three additional reports:

- The *Project Health Report*
- The *Test Progress History Report*
- The *Test Progress Report*

These reports are also described in the following sections.

9.1.2.1. The Project Overview Report

The *Project Overview Report* shows the main properties of a project listed in a table. The report shows:

- The number of defined systems under test
- The number of defined test environments
- The number of test cases
- The number of test suites
- The number of test case results
- The number of test suite results
- The average of the number of test case steps
- The average of the test case execution times

The only configurable property of the report is its display name.

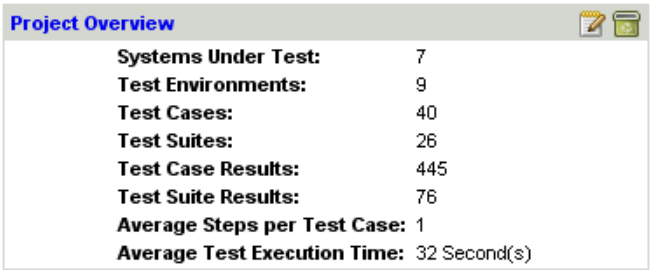


Figure 9.5. The Project Overview Report

9.1.2.2. The Latest Success Rate Report

The *Latest Success Rate* report shows the ratio of the newest test runs for a selected combination of system under test and test environment.

The configurable properties of the report are:

- The display name of the report.
- A boolean value indicating that this report always uses the active project (Klaros-Testmanagement Enterprise Edition only).
- Alternatively, the fixed project that this report will display.
- A boolean value indicating that this report always uses the active iteration (if available).
- Alternatively, the fixed iteration that this report will display (if available).
- The system under test to display.
- The test environment to display.

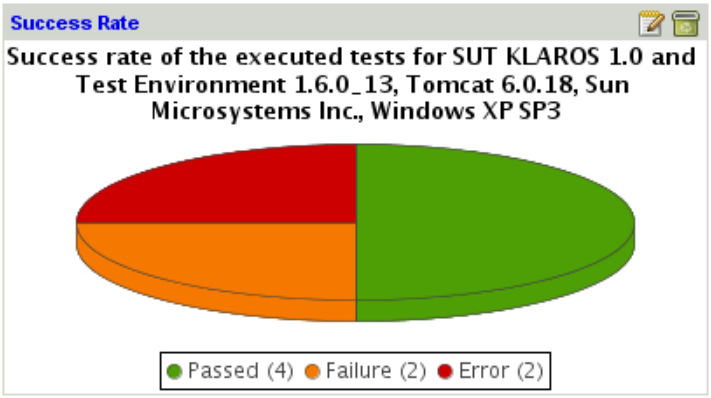

Success Rate
Success rate of the executed tests for SUT KLAROS 1.0 and Test Environment 1.6.0_13, Tomcat 6.0.18, Sun Microsystems Inc., Windows XP SP3
Passed (4)
Failure (2)
Error (2)

Figure 9.6. The Latest Success Rate Report

9.1.2.3. The Test Activity Report

The *Test Activity Report* shows the count and the results of the test runs for a selected combination of system under test and test environment in a selected period of time as a histogram.

The configurable properties of the report are:

- The name of the report.
- A boolean value indicating that this report always uses the active project (Klaros-Testmanagement Enterprise Edition only).
- Alternatively, the fixed project that this report will display.
- A boolean value indicating that this report always uses the active iteration (if available).
- Alternatively, the fixed iteration that this report will display (if available).
- The system under test to display.

- The test environment to display.
- The time period in days.

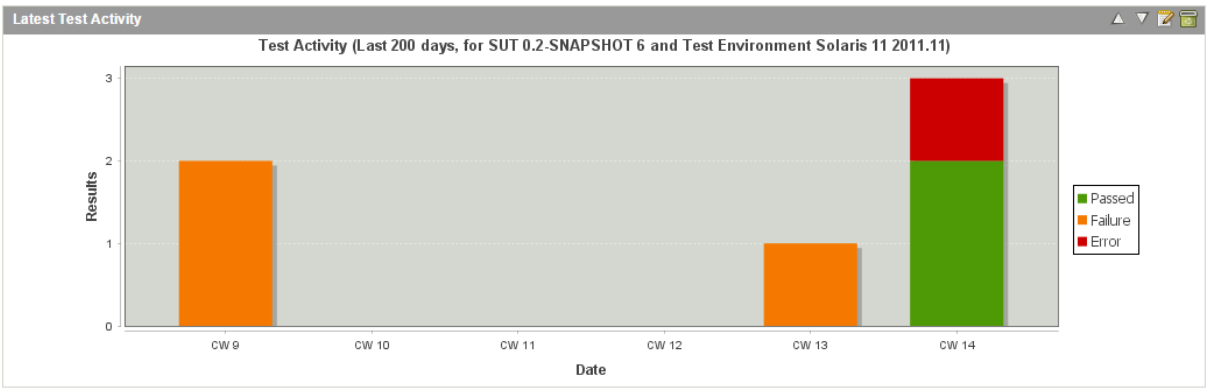


Figure 9.7. The Test Activity Report

9.1.2.4. The Project Health Matrix Report

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

The *Project Health Matrix Report* shows the project health as weather icons for each defined test environment and system under test in a matrix. The icons are calculated from the execution and success rate in each category which is shown in detail in a tool tip of each icon.

The configurable properties of the report are:

- The name of the report.
- A boolean value indicating that this report always uses the active project.
- A boolean value indicating that this report always uses the active iteration (if available).
- Alternatively, the fixed iteration that this report will display (if available).
- Alternatively, the fixed project that this report will display.
- The execution rates assigned to the different health categories.
- The success rates assigned to the different health categories.

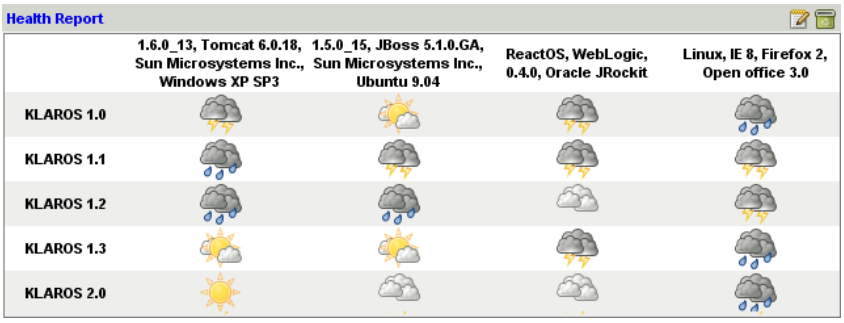

Health Report
1.6.0_13, Tomcat 6.0.18, Sun Microsystems Inc., Windows XP SP3
1.5.0_15, JBoss 5.1.0.GA, Sun Microsystems Inc., Ubuntu 9.04
ReactOS, WebLogic, 0.4.0, Oracle JRockit
Linux, IE 8, Firefox 2, Open office 3.0
KLAROS 1.0
KLAROS 1.1
KLAROS 1.2
KLAROS 1.3
KLAROS 2.0

Figure 9.8. The Project Health Matrix Report

9.1.2.5. The Test Progress Report

Feature only available in Klaros-Testmanagement Enterprise Edition

The *Test Progress Report* shows the rate of executed vs. defined tests of a project for a given test environment and a system under test.

The configurable properties of the report are:

- The name of the report.
- A boolean value indicating that this report always uses the active project.
- Alternatively, the fixed project that this report will display.
- A boolean value indicating that this report always uses the active iteration (if available).
- Alternatively, the fixed iteration that this report will display (if available).
- The system under test to display.
- The test environment to display.

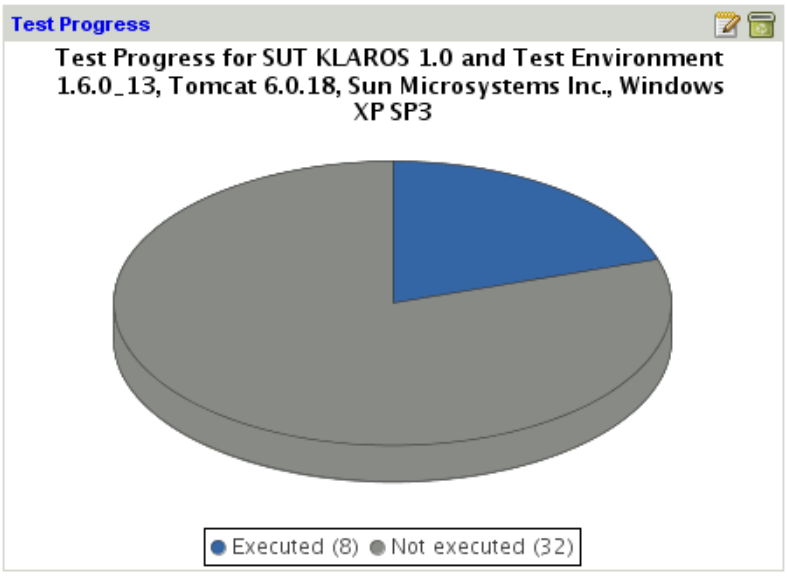

Test Progress
Test Progress for SUT KLAROS 1.0 and Test Environment 1.6.0_13, Tomcat 6.0.18, Sun Microsystems Inc., Windows XP SP3
Executed (8)
Not executed (32)

Figure 9.9. The Test Progress Report

9.1.2.6. The Test Progress History Report

Feature only available in Klaros-Testmanagement Enterprise Edition

The *Test Progress History Report* shows the rate of executed vs. defined tests of a project for a given test environment, system under test and time period.

The configurable properties of the report are:

- The name of the report.
- A boolean value indicating that this report always uses the active project.
- Alternatively, the fixed project that this report will display.
- A boolean value indicating that this report always uses the active iteration (if available).
- Alternatively, the fixed iteration that this report will display (if available).
- The system under test to display.
- The test environment to display.
- The time period of the report.

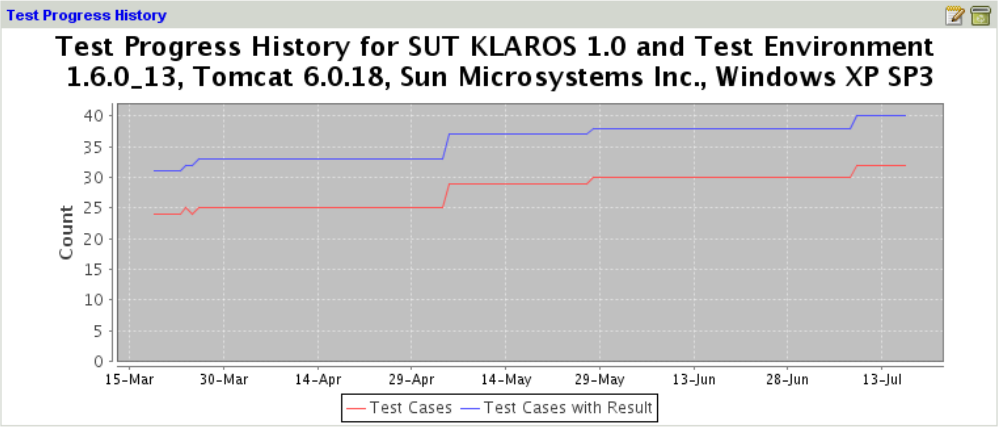


Figure 9.10. The Test Progress History Report

9.2. Report Templates

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

The *Report Templates* screen allows to generate PDFs of user-defined reports. The icon starts the generation and, if necessary, opens a new window to enter the needed parameters. The process of creating user-defined reports is explained in Section 10.2, “Report Templates”.

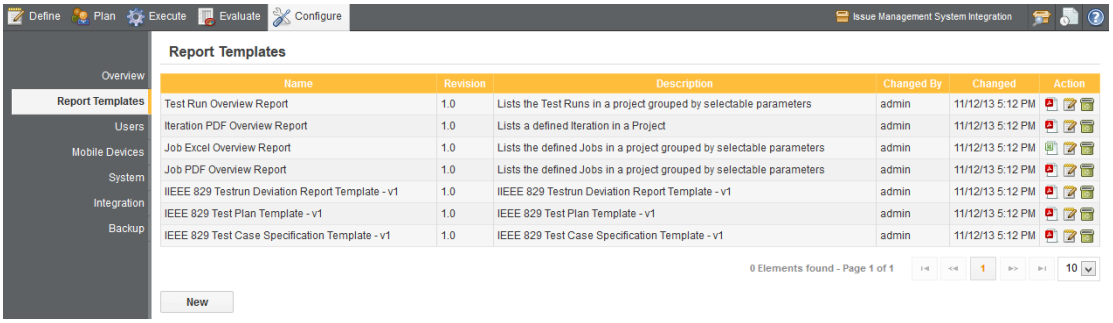


Figure 9.11. User Defined Reports Screens

If the user-defined report is parameterized, all required parameters must be entered to start the generation of the report.

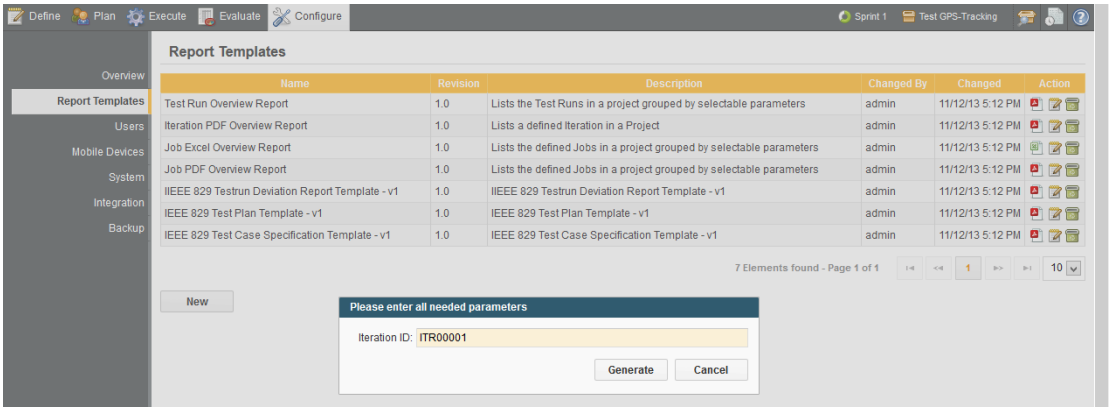


Figure 9.12. Generate a parameterized Report

Note

The **Generate** button becomes active after entering all required parameters.

9.3. Test Run History Report

On the test run history screen, a report for the test runs using the selected system under test can be generated. The report contains a summary, with two result history graphics (percentage and total number), and the details of each test run (Figure 9.13).

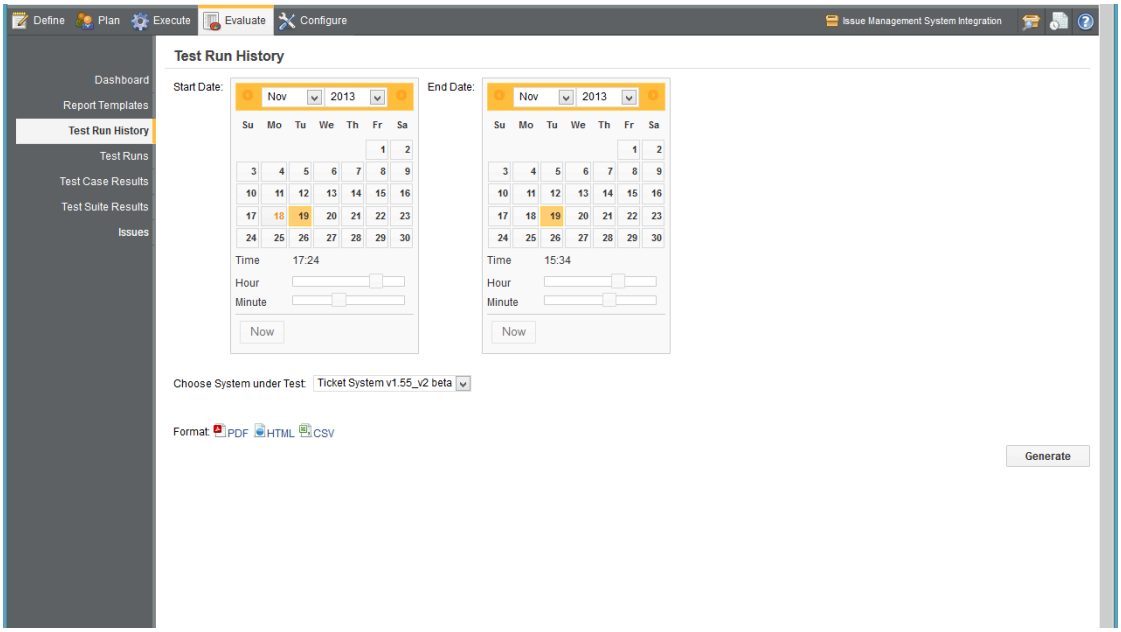
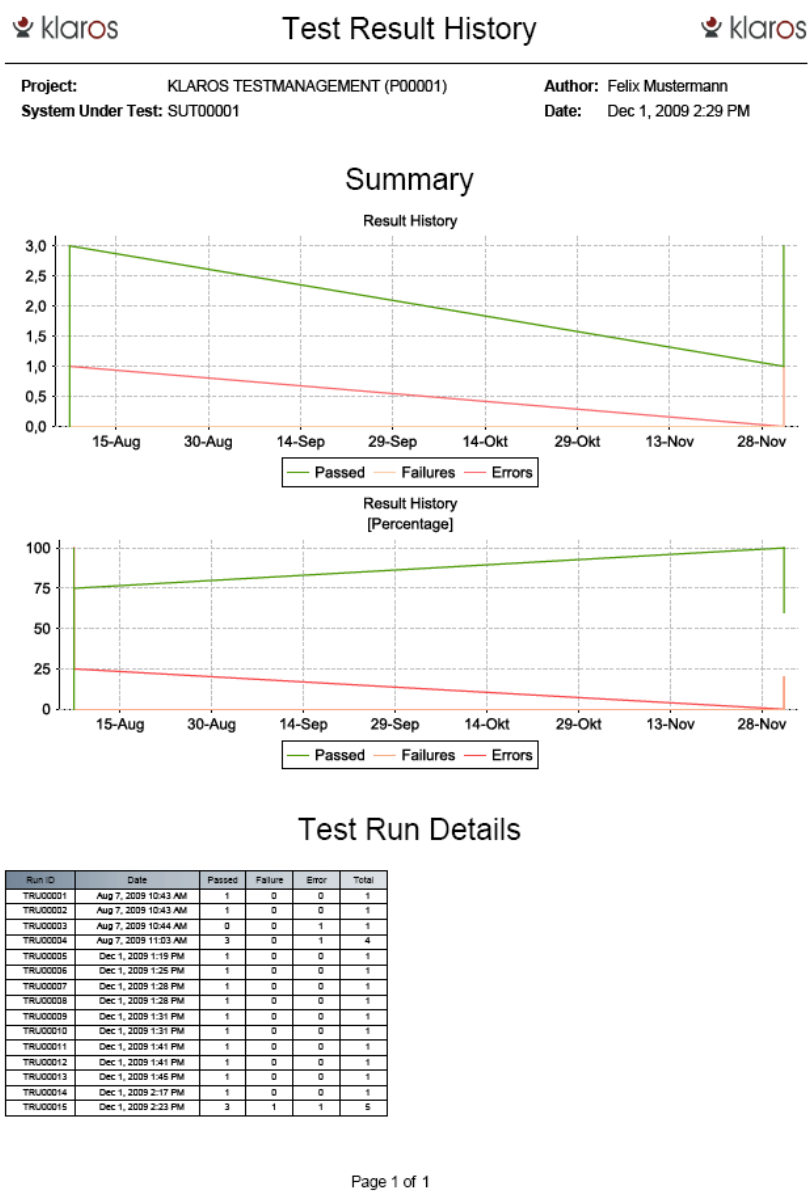


Figure 9.13. The Test Run History Report

Klaros-Testmanagement can export these test run report in three different file formats:

- PDF
- HTML
- CSV

After selecting the *Start Date*, *End Date* and *System under Test* the **Generate** can be clicked, enabling the user to choose their preferred file format in which to download the report (see Figure 9.13). The layout of the report is shown in Figure 9.14.



Run ID	Date	Passed	Failure	Error	Total
TRU00001	Aug 7, 2009 10:43 AM	1	0	0	1
TRU00002	Aug 7, 2009 10:43 AM	1	0	0	1
TRU00003	Aug 7, 2009 10:44 AM	0	0	1	1
TRU00004	Aug 7, 2009 11:03 AM	3	0	1	4
TRU00005	Dec 1, 2009 1:19 PM	1	0	0	1
TRU00006	Dec 1, 2009 1:25 PM	1	0	0	1
TRU00007	Dec 1, 2009 1:28 PM	1	0	0	1
TRU00008	Dec 1, 2009 1:28 PM	1	0	0	1
TRU00009	Dec 1, 2009 1:31 PM	1	0	0	1
TRU00010	Dec 1, 2009 1:31 PM	1	0	0	1
TRU00011	Dec 1, 2009 1:41 PM	1	0	0	1
TRU00012	Dec 1, 2009 1:41 PM	1	0	0	1
TRU00013	Dec 1, 2009 1:45 PM	1	0	0	1
TRU00014	Dec 1, 2009 2:17 PM	1	0	0	1
TRU00015	Dec 1, 2009 2:23 PM	3	1	1	5

Figure 9.14. The Test Run History Report Layout

9.4. Test Runs

The *Test Runs* screen allows the user to generate reports for executed test runs (Figure 9.15).

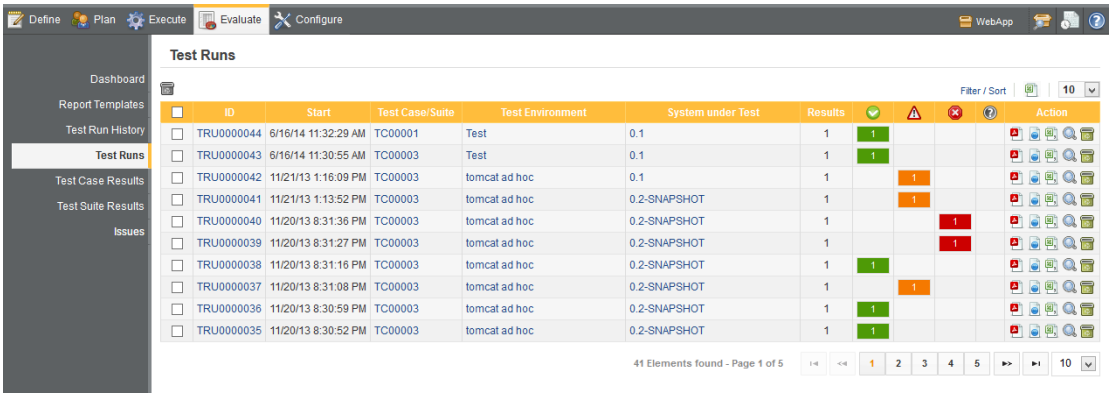

Define
Plan
Execute
Evaluate
Configure
WebApp
Test Runs
Dashboard
Report Templates
Test Run History
Test Runs
Test Case Results
Test Suite Results
Issues
Filter / Sort
10
ID
Start
Test Case/Suite
Test Environment
System under Test
Results
Action
TRU0000044 6/16/14 11:32:29 AM TC00001 Test 0.1 1
TRU0000043 6/16/14 11:30:55 AM TC00003 Test 0.1 1
TRU0000042 11/21/13 1:16:09 PM TC00003 tomcat ad hoc 0.1 1
TRU0000041 11/21/13 1:13:52 PM TC00003 tomcat ad hoc 0.2-SNAPSHOT 1
TRU0000040 11/20/13 8:31:36 PM TC00003 tomcat ad hoc 0.2-SNAPSHOT 1
TRU0000039 11/20/13 8:31:27 PM TC00003 tomcat ad hoc 0.2-SNAPSHOT 1
TRU0000038 11/20/13 8:31:16 PM TC00003 tomcat ad hoc 0.2-SNAPSHOT 1
TRU0000037 11/20/13 8:31:08 PM TC00003 tomcat ad hoc 0.2-SNAPSHOT 1
TRU0000036 11/20/13 8:30:59 PM TC00003 tomcat ad hoc 0.2-SNAPSHOT 1
TRU0000035 11/20/13 8:30:52 PM TC00003 tomcat ad hoc 0.2-SNAPSHOT 1
41 Elements found - Page 1 of 5
1 2 3 4 5
10

Figure 9.15. The Test Runs Screen

This page lists all test runs that have been executed in the selected project. To get a better overview of the test runs, the sorting and filtering functionality of the *Filter / Sort* panel can be used, as described in Section 5.2.5, “Filtering and Sorting”.

Pressing the delete icon removes the test run.

Note

Deleting a test run will delete corresponding test case results and test suite results.

Clicking the icon displays the details of the test run.

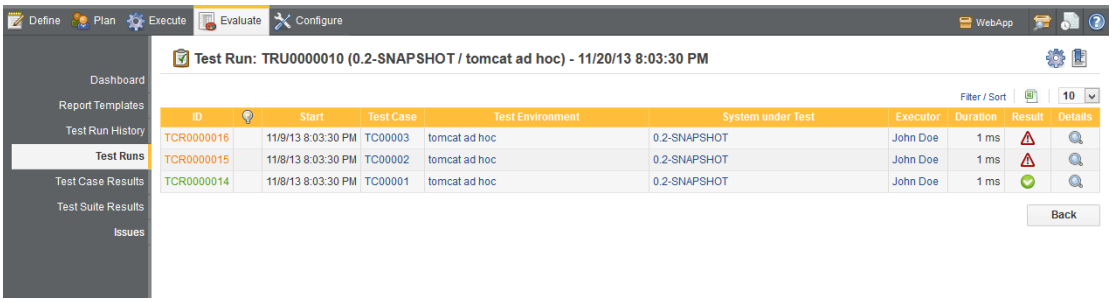


Figure 9.16. The Test Run Details Screen

mobile edition Only available in Klaros-Testmanagement Mobile Edition.

If a test run has been executed on a mobile device, an info icon will be displayed in the info column. Pressing this icon will open up the *Mobile Device Details* page of the mobile device (see Section 10.4, “Mobile Devices”).

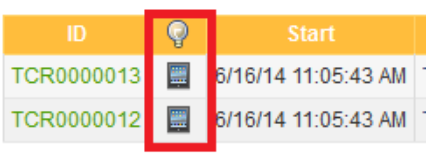


Figure 9.17. The Test Run Info Column

9.4.1. The Test Run Report

This type of report is generated in PDF format and its layout is shown in Figure 9.18. It details the date of execution of a test run and its result, specifying the test cases executed and whether they were successful, failed or ended in error. Additionally a pie chart summarizes the result is shown.

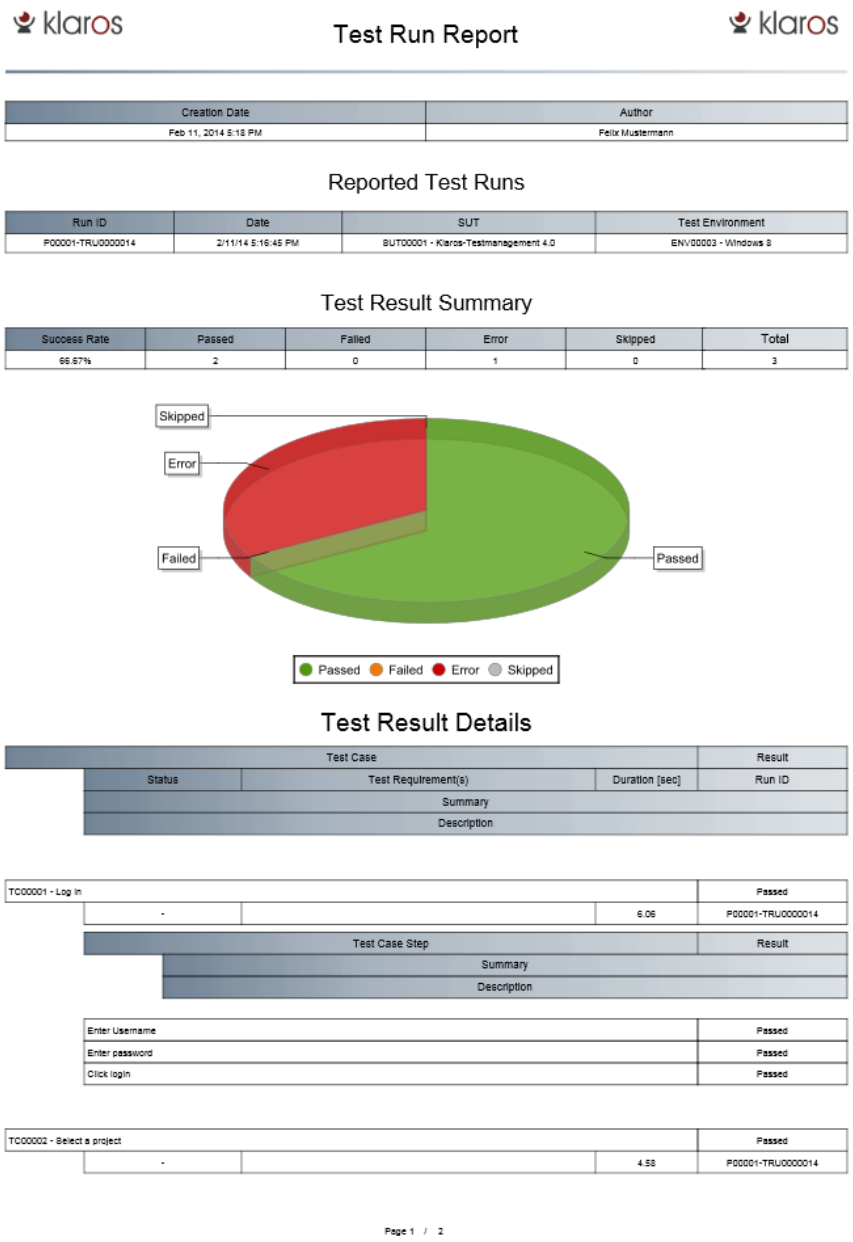


Figure 9.18. The Single Test Run Report

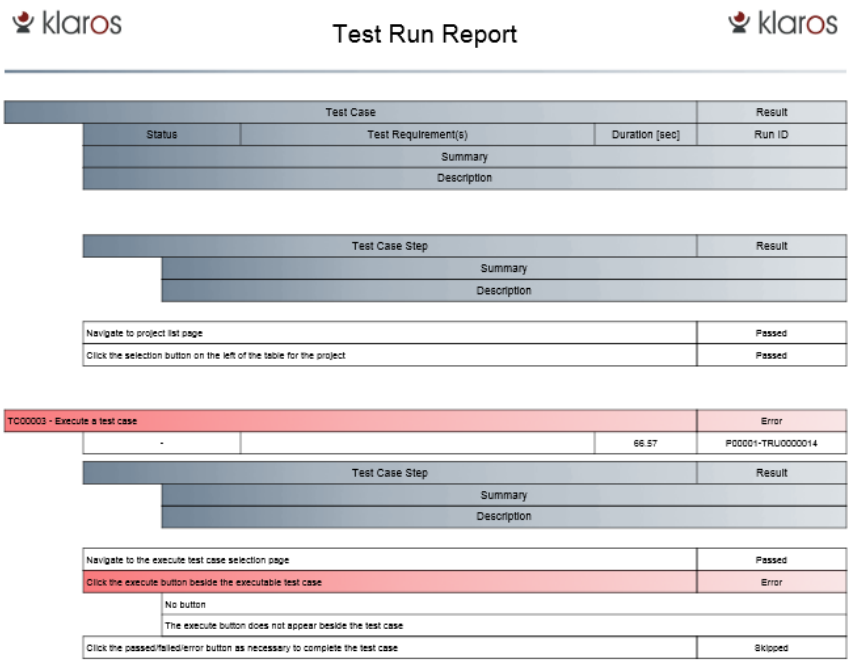


Figure 9.19. The Single Test Run Report (continued)

9.5. Test Case Results

The *Test Case Result* screen shows results for individual test cases. The total number of passed, failed, error and skipped results for each test case is shown (Figure 9.20). To get a better overview of the test cases, the sorting and filtering functionality of the *Filter / Sort* panel can be used, as described in Section 5.2.5, “Filtering and Sorting”.

This screen displays an overview of the test cases for the project. To narrow the search, the filtering and sorting functionality of the page can be used. The latest result of a test case is displayed in the column.

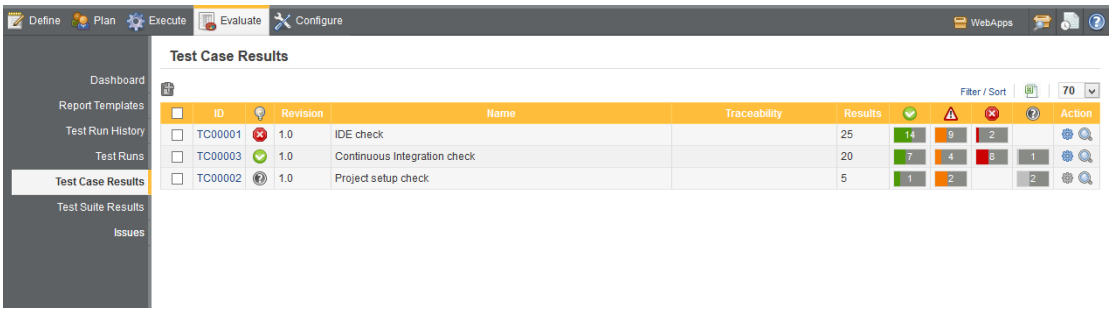


Figure 9.20. The Test Case Results Screen

Clicking the icon starts a retest of the corresponding test case and clicking the opens the test case result page for the corresponding test case.

The icon opens a dialog which enables the user to create a test suite from the selected test cases. The dialog to create the new test suite is shown in Figure 9.21.

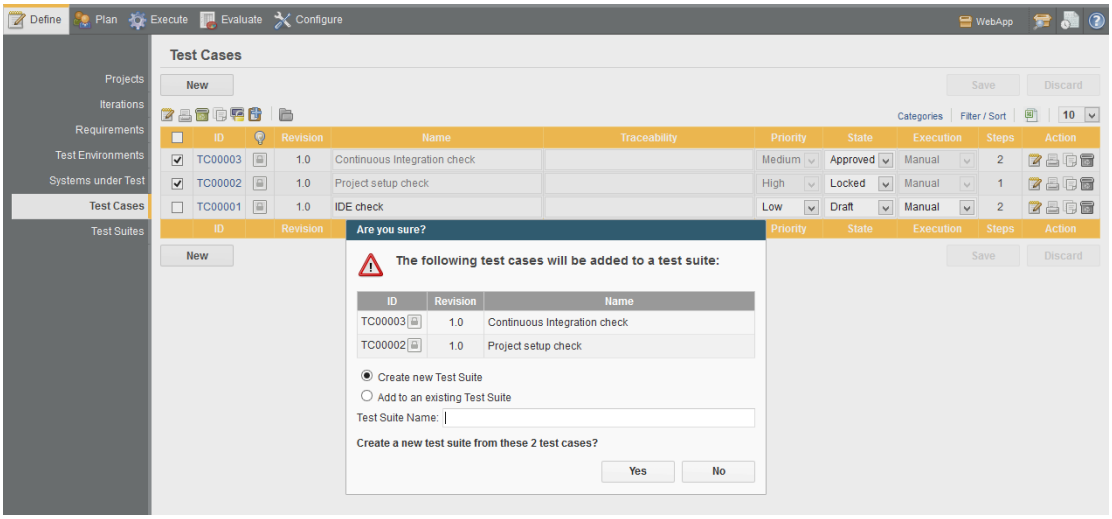


Figure 9.21. The Test Suite from Test Case Results Pop-up

9.5.1. Single Test Case Results

This page shows individual executions of the selected test case. The result of each execution is shown, as well as the system under test and test environment used (Figure 9.22).

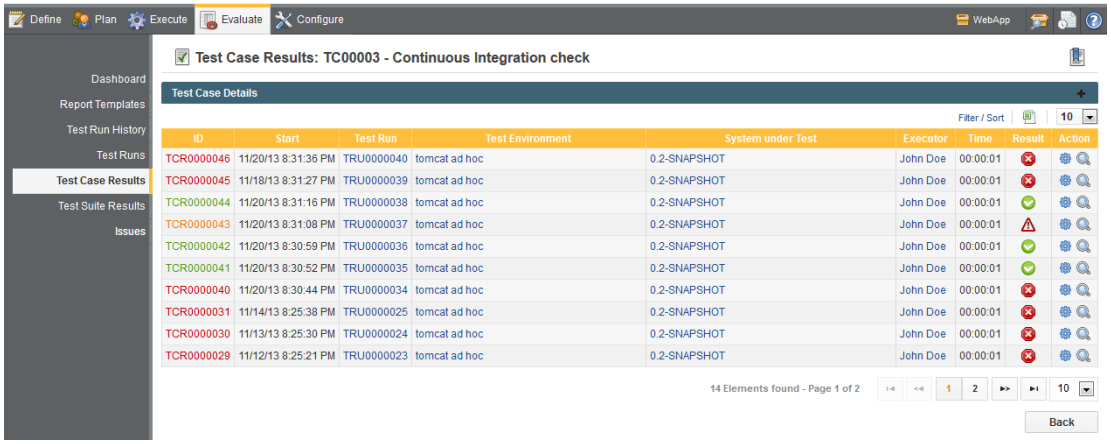

Define
Plan
Execute
Evaluate
Configure
WebApp
Test Case Results: TC00003 - Continuous Integration check
Dashboard
Report Templates
Test Run History
Test Runs
Test Case Results
Test Suite Results
Issues
Test Case Details
Filter / Sort
10
ID
Start
Test Run
Test Environment
System under Test
Executor
Time
Result
Action
TCR0000046 11/20/13 8:31:36 PM TRU0000040 tomcat ad hoc 0.2-SNAPSHOT John Doe 00:00:01
TCR0000045 11/18/13 8:31:27 PM TRU0000039 tomcat ad hoc 0.2-SNAPSHOT John Doe 00:00:01
TCR0000044 11/20/13 8:31:16 PM TRU0000038 tomcat ad hoc 0.2-SNAPSHOT John Doe 00:00:01
TCR0000043 11/20/13 8:31:08 PM TRU0000037 tomcat ad hoc 0.2-SNAPSHOT John Doe 00:00:01
TCR0000042 11/20/13 8:30:59 PM TRU0000036 tomcat ad hoc 0.2-SNAPSHOT John Doe 00:00:01
TCR0000041 11/20/13 8:30:52 PM TRU0000035 tomcat ad hoc 0.2-SNAPSHOT John Doe 00:00:01
TCR0000040 11/20/13 8:30:44 PM TRU0000034 tomcat ad hoc 0.2-SNAPSHOT John Doe 00:00:01
TCR0000031 11/14/13 8:25:38 PM TRU0000025 tomcat ad hoc 0.2-SNAPSHOT John Doe 00:00:01
TCR0000030 11/13/13 8:25:30 PM TRU0000024 tomcat ad hoc 0.2-SNAPSHOT John Doe 00:00:01
TCR0000029 11/12/13 8:25:21 PM TRU0000023 tomcat ad hoc 0.2-SNAPSHOT John Doe 00:00:01
14 Elements found - Page 1 of 2
1
2
10
Back

Figure 9.22. The Single Test Case Results Screen

The test case details panel in the page shows all details of the selected test case. The result of the test cases run is shown using the following icons:

- the test finished successfully
- a failure of the system under test has been detected
- an error occured during the test execution
- the test result could not be determined

In addition, the ID of each test case result is colored green for passed, orange for failed, red for error or gray for skipped.

If the icon appears in the ID column, this means that the corresponding test case result has been manually changed by a user.

Clicking the icon starts a retest of the test case.

To get a better overview of the test case results, the sorting and filtering functionality of the *Filter / Sort* panel can be used, as described in Section 5.2.5, “Filtering and Sorting”.

9.5.1.1. Navigating back to the Overview Page

Clicking the **Back** button brings the user back to the test case results screen (Figure 9.20), where they may choose another test case.

Clicking on the icon opens up the test case result details screen for the corresponding test case result (Figure 9.23).

9.5.1.2. The Test Case Results Details Screen

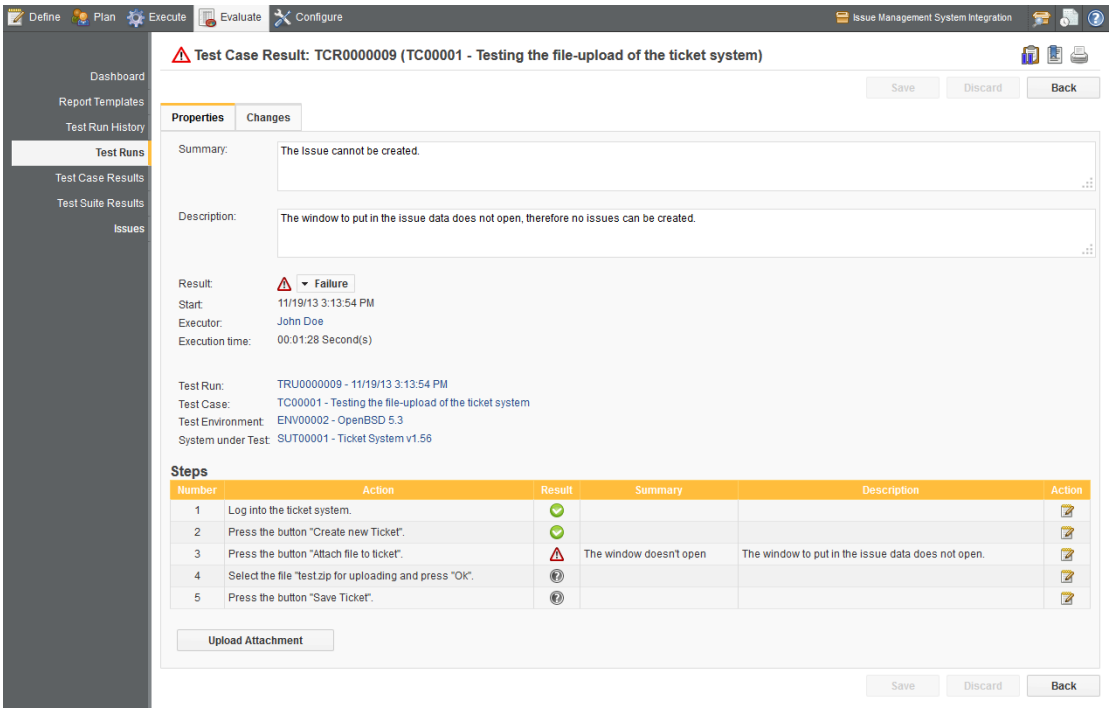


Figure 9.23. The Test Case Results Details Screen

This page shows the individual step results of the test case result. Users may also edit the *Summary*, *Description* and *Result* of the test case result and of each individual step result. All changes are logged, and any change to results is cascaded to the test case result, test suite result and test run. If a test case result has been changed, the icon is displayed on this page and in all tables where the result is displayed. The *Result* field of a test case result may only be changed in Klaros-Testmanagement Enterprise Edition. Figure 9.24 shows the individual step edit dialog.

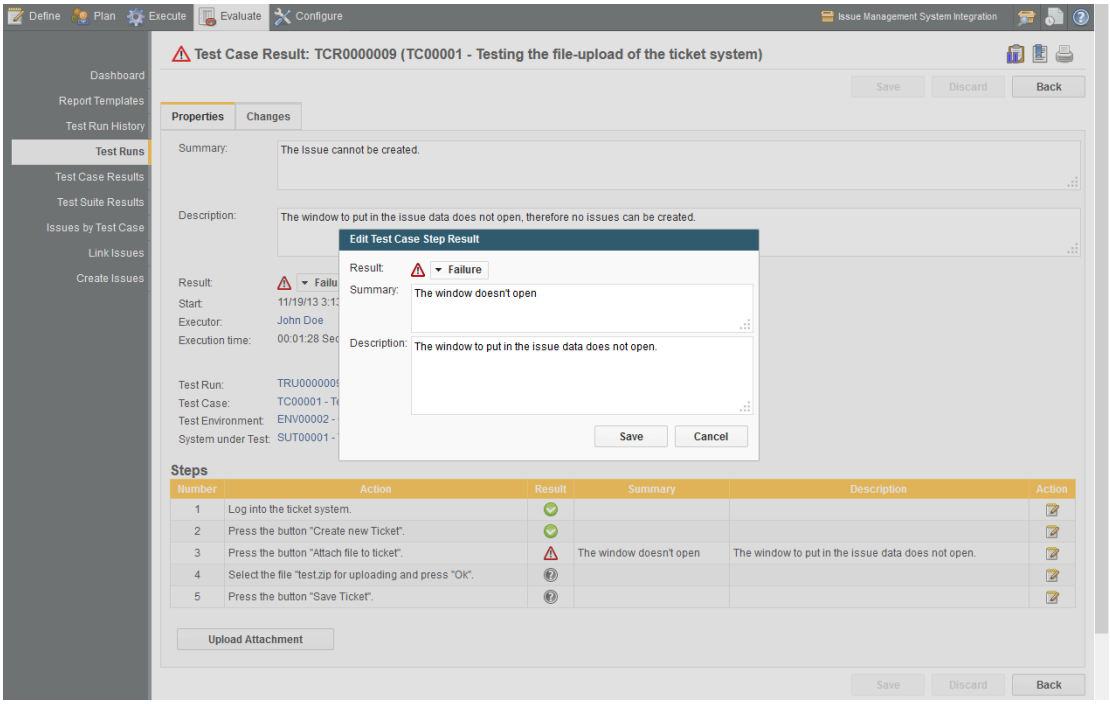
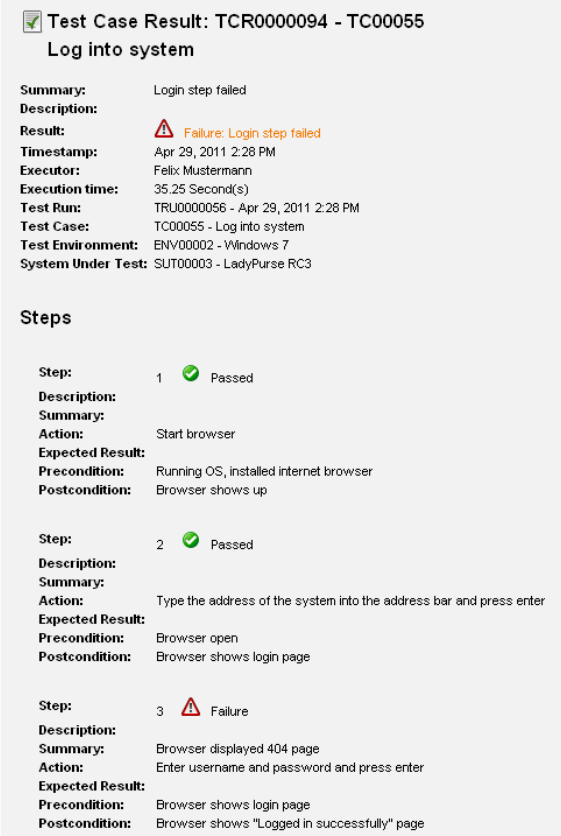


Figure 9.24. Editing Test Case Step Results

Clicking the icon starts a retest of the test case.

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

This screen can also be viewed as a printer-friendly page (Figure 9.25).



Test Case Result: TCR0000094 - TC00055
Log into system

Summary:	Login step failed
Description:	
Result:	Failure: Login step failed
Timestamp:	Apr 29, 2011 2:28 PM
Executor:	Felix Mustermann
Execution time:	35.25 Second(s)
Test Run:	TRU0000056 - Apr 29, 2011 2:28 PM
Test Case:	TC00055 - Log into system
Test Environment:	ENV00002 - Windows 7
System Under Test:	SUT00003 - LadyPurse RC3

Steps

Step:	1 Passed
Description:	
Summary:	
Action:	Start browser
Expected Result:	
Precondition:	Running OS, installed internet browser
Postcondition:	Browser shows up

Step:	2 Passed
Description:	
Summary:	
Action:	Type the address of the system into the address bar and press enter
Expected Result:	
Precondition:	Browser open
Postcondition:	Browser shows login page

Step:	3 Failure
Description:	
Summary:	Browser displayed 404 page
Action:	Enter username and password and press enter
Expected Result:	
Precondition:	Browser shows login page
Postcondition:	Browser shows "Logged in successfully" page

Figure 9.25. The Test Case Results Print Screen

9.5.1.2.1. Location

Only available in Klaros-Testmanagement Mobile Edition.

If location tracking has been enabled on the Klaros-Testmanagement Mobile Edition Android Client this tab displays an interactive map. for every test case step result there is a marker displayed on the map. The position of the marker on the map corresponds to the coordinates that have been tracked when the user entered the test case step result on his mobile device.

Clicking on one of the markers opens up a popup with details about the test case step result, like the verdict or the description the user has entered.

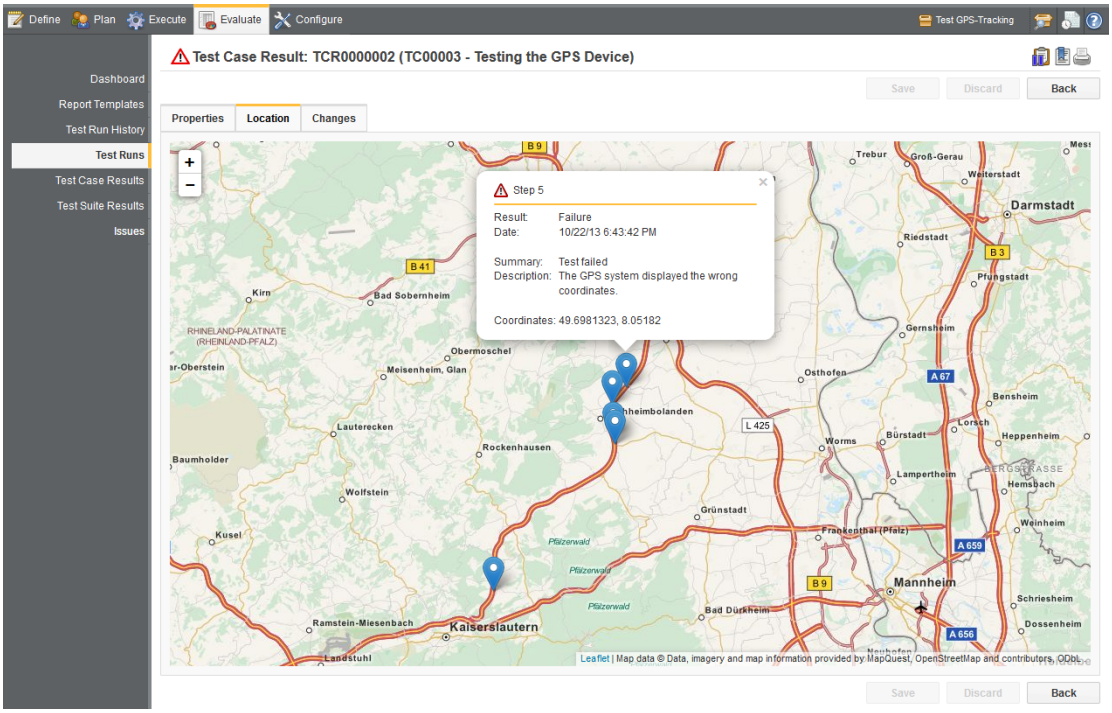


Figure 9.26. The Location Tab

The map has several interactive elements on it. The map can be moved in any direction by pressing the left mouse button, holding the mouse button and then moving the mouse. Initially, the map is centered to the marker of the first test case step result. The map is also scaleable. The scaling can be triggered by scrolling the mouse wheel up (scale up) or down (scale down). Alternatively, the toolbar in the upper left of the map can be used to scale the map.

9.6. Test Suite Results

The test suite result screen shows results for individual test suites. The total number of results for each test suite is shown (Figure 9.27). To get a better overview of the test suites, the sorting and filtering functionality of the *Filter / Sort* panel can be used, as described in Section 5.2.5, “Filtering and Sorting”.

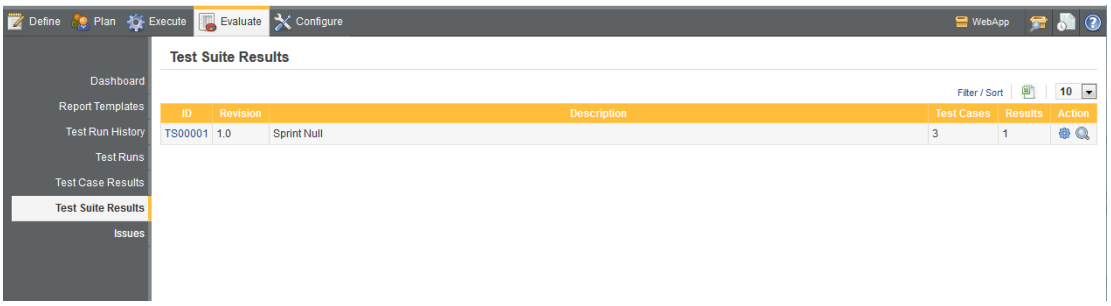


Figure 9.27. The Test Suite Results Screen

Clicking the icon starts a retest of the test suite and clicking the icon opens the *Single Test Suite Results* page.

9.6.1. Single Test Suite Results

This page shows individual executions of the selected test case. The result of each execution is shown, as well as the system under test and test environment used. (Figure 9.28).

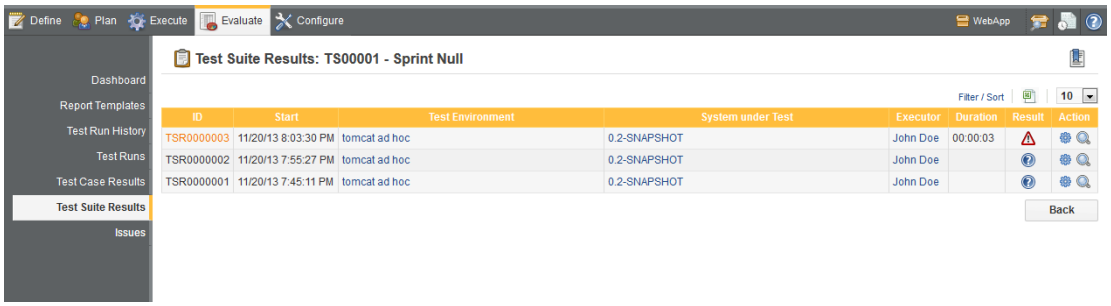


Figure 9.28. The Test Suite Results Screen

To get a better overview of the test suite results, the sorting and filtering functionality of the *Filter / Sort* panel can be used, as described in Section 5.2.5, "Filtering and Sorting".

Clicking the opens the *Test Suite Result Details* page for the corresponding test suite result (Figure 9.29).

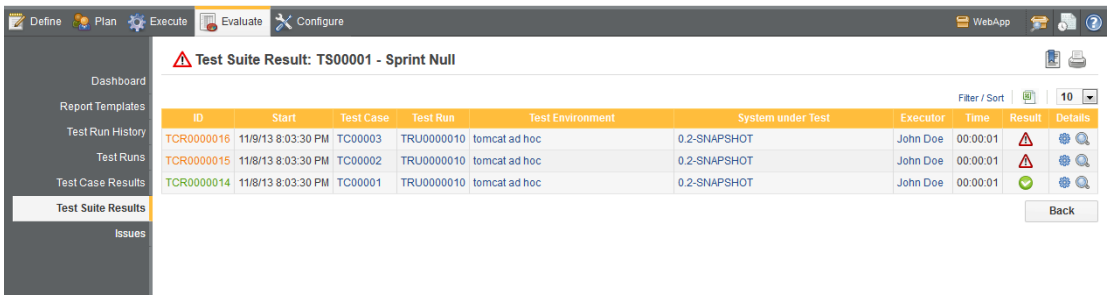


Figure 9.29. The Test Suite Results Screen - Test Results

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

This screen can also be viewed as a printer-friendly page (Figure 9.30).

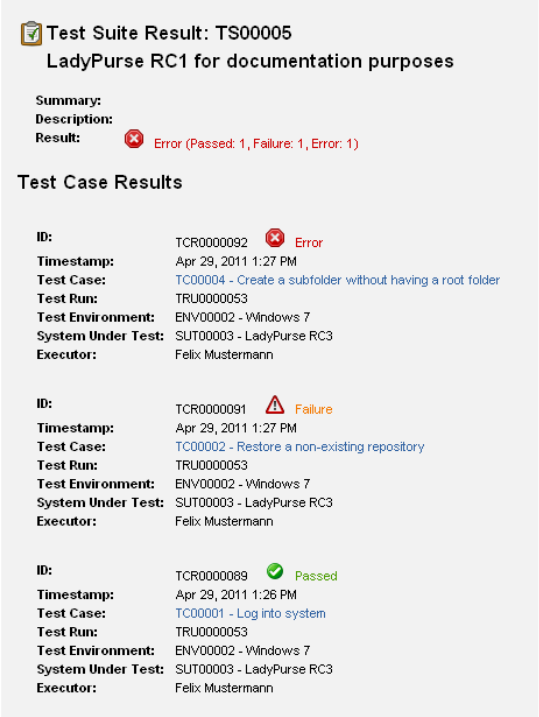

Test Suite Result: TS00005
LadyPurse RC1 for documentation purposes
Summary:
Description:
Result:
Error (Passed: 1, Failure: 1, Error: 1)
Test Case Results
ID: TCR0000092 Error
Timestamp: Apr 29, 2011 1:27 PM
Test Case: TC00004 - Create a subfolder without having a root folder
Test Run: TRU0000053
Test Environment: ENV00002 - Windows 7
System Under Test: SUT00003 - LadyPurse RC3
Executor: Felix Mustermann
ID: TCR0000091 Failure
Timestamp: Apr 29, 2011 1:27 PM
Test Case: TC00002 - Restore a non-existing repository
Test Run: TRU0000053
Test Environment: ENV00002 - Windows 7
System Under Test: SUT00003 - LadyPurse RC3
Executor: Felix Mustermann
ID: TCR0000089 Passed
Timestamp: Apr 29, 2011 1:26 PM
Test Case: TC00001 - Log into system
Test Run: TRU0000053
Test Environment: ENV00002 - Windows 7
System Under Test: SUT00003 - LadyPurse RC3
Executor: Felix Mustermann

Figure 9.30. The Test Suite Results Screen - Print Test Results

The results of the test suites are shown using the following icons:

- The test finished successfully
- A failure of the system under test has been detected
- An error occured during the test execution
- The test result could not be determined

Clicking the icon starts a retest of the test suite.

9.7. Issues

With Klaros-Testmanagement it is possible to link issues from external issue management systems (like JIRA or Bugzilla) with test cases. In addition to this, issues can be created from within Klaros-Testmanagement and even during test execution.

9.7.1. Issues by Test Case

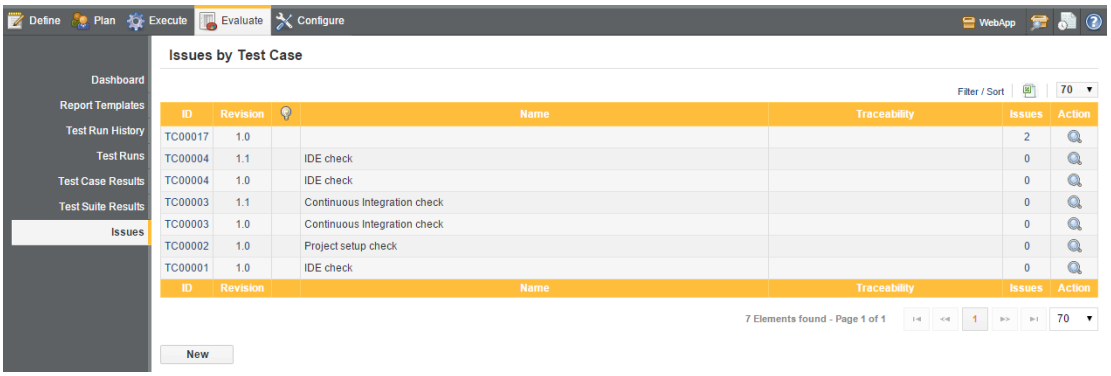


Figure 9.31. The Issues by Test Case Screen

This page shows a list of all test cases in every revision and the number of issues linked to it. Pressing the icon opens up a separate page (see Section 9.7.2.2, “Issues by Test Case Details”) where the individual linked issues are listed. Pressing the *New* button opens up a new page in where an issue can be created in any of the configured issue management systems and then linked to a test case.

9.7.2. Issue Details (Creating a new Issue)

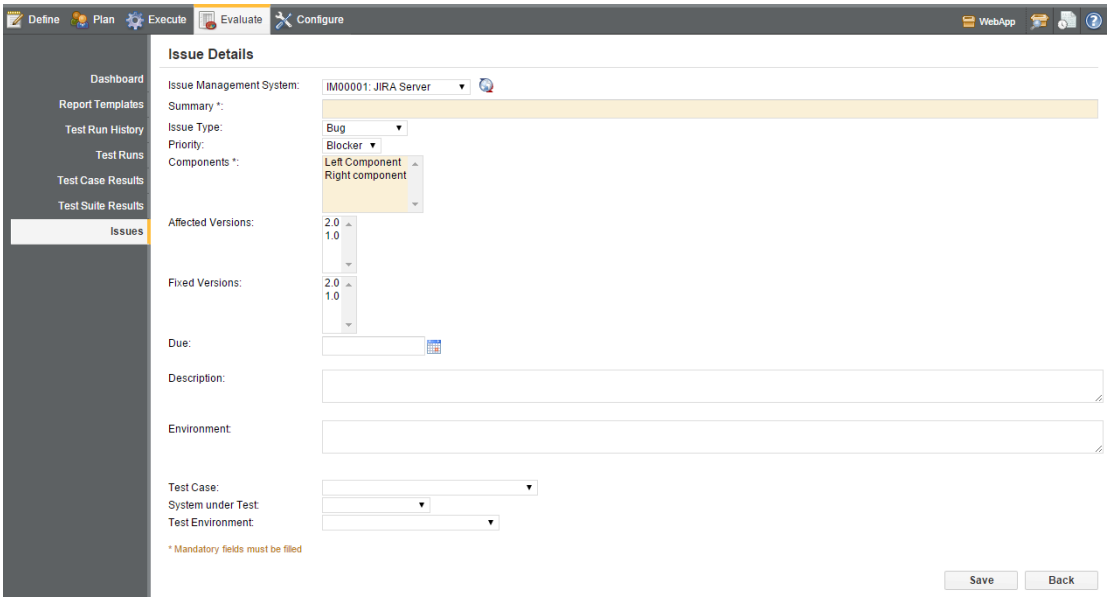


Figure 9.32. The Create Issue Page

Upon entering this page at first an issue management system has to be selected from the *Issue Management System* dropdown list.

Note

If the list of issue management systems is empty, then at least one issue management system has to be configured and also available in the selected project. The configuration of an issue management system is described in Section 10.6.1, "Issue Management", and Section 6.1.6.1, "Editing the Issue Management Systems in a Project" contains instructions on how to add an issue management system to a project.

By default the system uses the credentials entered at the login screen to authenticate the user against the issue management system. In a Bugzilla issue management system the email/password combination of the user is tried as well, as this is the common case for a login name here. If the credentials entered at login do not work and the issue management system has not yet been used during the session, Klaros-Testmanagement will request new authentication information in a dialog window.

Pressing the icon opens up the external issue management system in a new tab.

After the desired issue management system has been selected, a list of issue fields is shown. The exact number and naming of those fields varies depending on the type and configuration of the issue management system. Mandatory fields are shaded in a different color to emphasize their importance.

Note

All mandatory fields need to be filled, otherwise the issue cannot be saved.

Issue Fields for every Issue Management System

- Test Case

  The test case with which the issue should be linked.

- System under Test

  The system under test with which the issue should be linked.

- Update Test Case State?

  Selecting this checkbox opens up a new dropdown list in which the new state of the linked test case can be selected.

9.7.2.1. Issue Details (Editing an Existing Issue)

Editing an issue is done in the same page as creating as Section 9.7.2, "Issue Details (Creating a new Issue)". The only difference is that a button *New* is placed on the lower left section of the page. This button serves as a convinience method when creating multiple issues in a row. Pressing this button opens up a new *Issue Details* page in which another issue can be configured.

Tip

After creating and saving a new issue, the *New* button also appears, so that another issue can be created right away!

9.7.2.2. Issues by Test Case Details

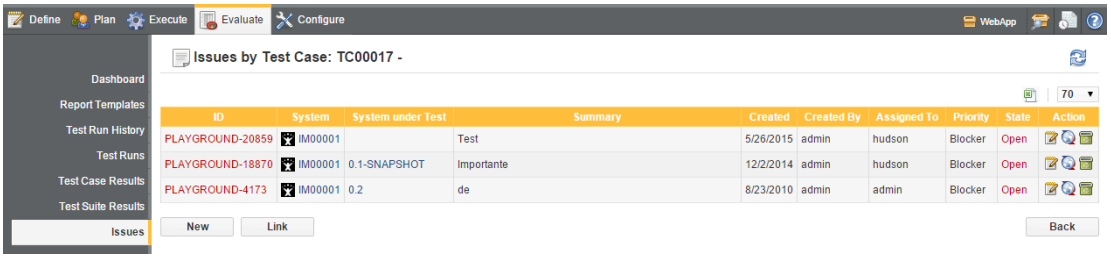


Figure 9.33. The Issues by Test Case Details Screen

The *Issues by Test Case Details* page shows all issues which are linked to the selected test case.

Pressing the icon opens up the Section 9.7.2.1, "Issue Details (Editing an Existing Issue)" page, where the individual fields of the issue can be edited.

Tip

It is possible to move an issue from one test case to another!

Pressing the icon opens up the issue in the appertaining issue management systems in a new tab.

Pressing the icon removes the issue from the test case.

Important

This does not delete the issue in the issue management system, only the reference to the test case in Klaros-Testmanagement!

Pressing the *New* button opens up the Section 9.7.2, "Issue Details (Creating a new Issue)" page.

Pressing the *Link* button opens up the Link Issues page, in which existing issues from issue management systems can be linked to this test case.

The *Issues by Test Case* page lists all test cases that have linked issues or results with linked issues (Figure 9.31, "The Issues by Test Case Screen"). Clicking the button opens the *Browse Issues* page for the corresponding test case (Figure 9.33, "The Issues by Test Case Details Screen").

The issue *ID* is a hyperlink to the corresponding issue in the issue management system.

9.7.3. Link Issues

While most issues get detected during execution of a test case, sometimes it is necessary to defer the creation of an issue in the issue management system, and some issues are raised on an ad hoc basis. In order to preserve a relation to test cases, it is possible to link issues created in an issue management system to test cases in Klaros-Testmanagement.

The *Link Issues* page lists all test cases in the currently active project. Clicking the icon opens the *Link Issues* page for the corresponding test case.

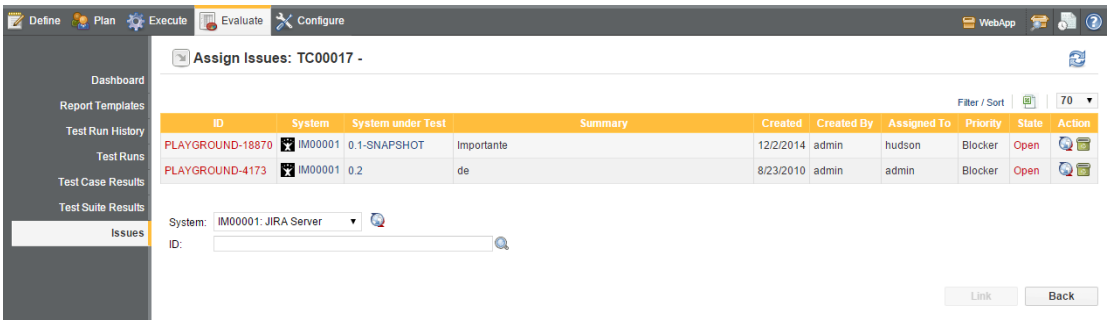


Figure 9.34. The Test Case Selection Screen

To link a new issue to the test case, an issue management system must be selected. Pressing the **Link** button causes the issue with the entered *ID* to be attached to the test case.

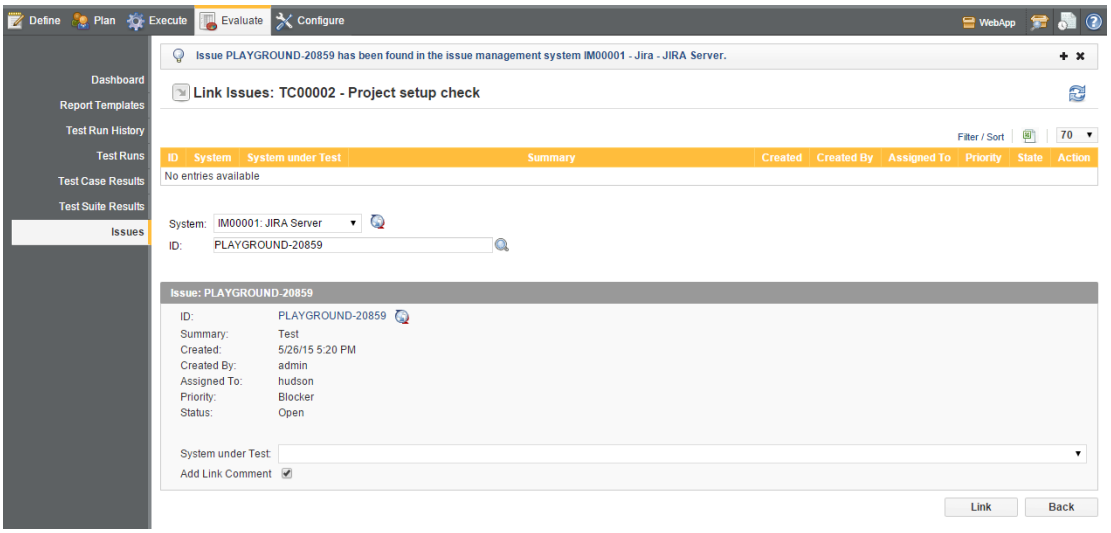


Figure 9.35. The Link Issues Screen

Pressing the icon opens up the issue in the appertaining issue management systems in a new tab.

Pressing the icon removes the issue from the test case.

Important

This does not delete the issue in the issue management system, only the reference to the test case in Klaros-Testmanagement!

- System

  The issue management system from which issues are to be linked.

- ID

  The id of the issue to be linked.

Pressing the icon searches the issue management system for an issue which matches the entered id. Upon a successful search, pressing the *Link* button links the issue to the test case.

9.8. Bookmarking Results

Klaros-Testmanagement allows the bookmarking of test run details, test case result details and test suite result details. The icon on these pages is a link to the persistent bookmark.

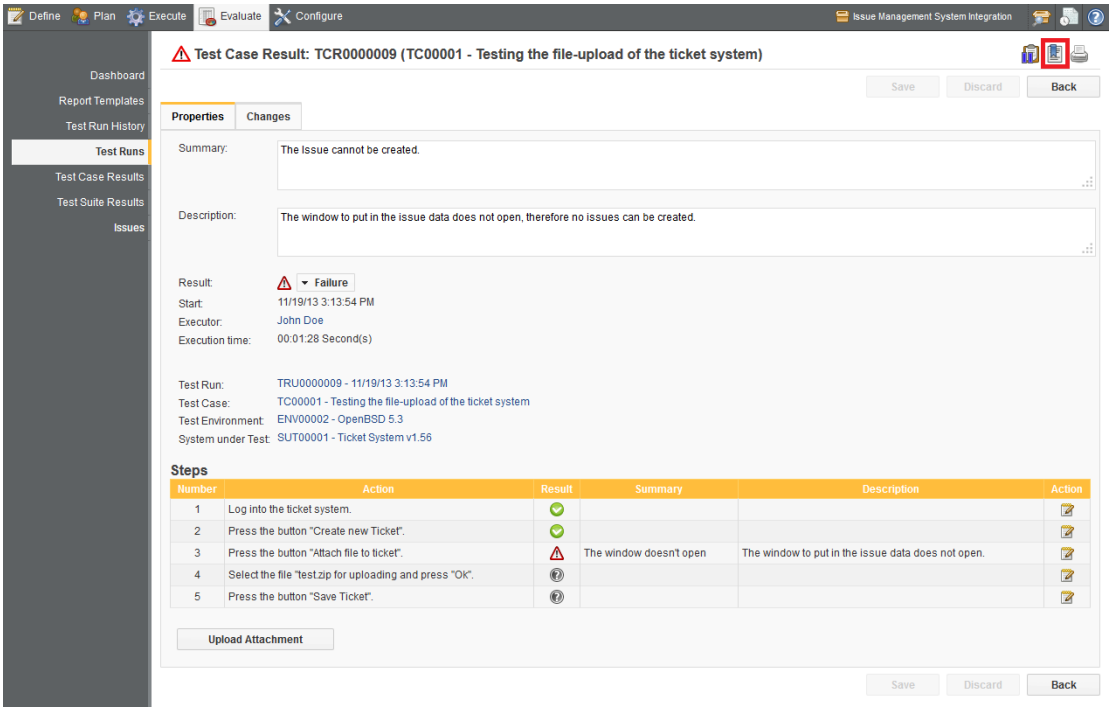


Figure 9.36. The Test Case Results Screen

Section 6.1.6.1, “Editing the Issue Management Systems in a Project” explains the configuration of controlling access to bookmarked pages in the related project.

Chapter 10. Configure

The *Configure* section provides administrative functions like managing custom reports, creating and editing user accounts, changing application settings, configuring remote system parameters, and backing up and restoring projects.

10.1. Overview

The main page of the *Configure* section is the *Overview* page. This page contains information about the operating system, Java runtime environment, memory usage and some additional information about the system as shown in Figure 10.1.

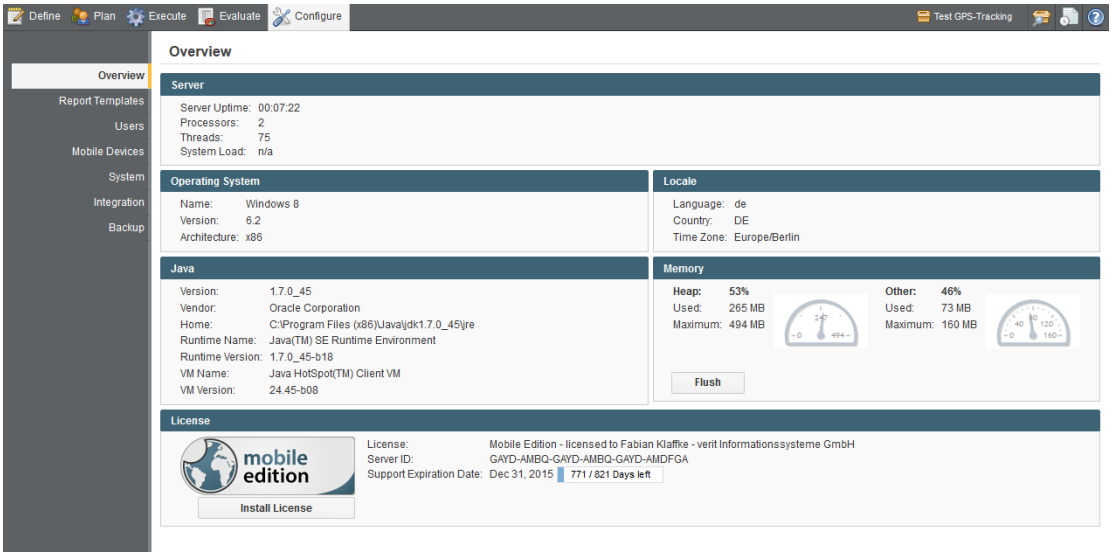


Figure 10.1. The Overview Page

10.2. Report Templates

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

On the *Report Templates* pages it is possible to create, edit, delete and generate user-defined reports.

Clicking the icon starts the creation of the corresponding report in the PDF format. Likewise the will start the creation of the report in Excel format.

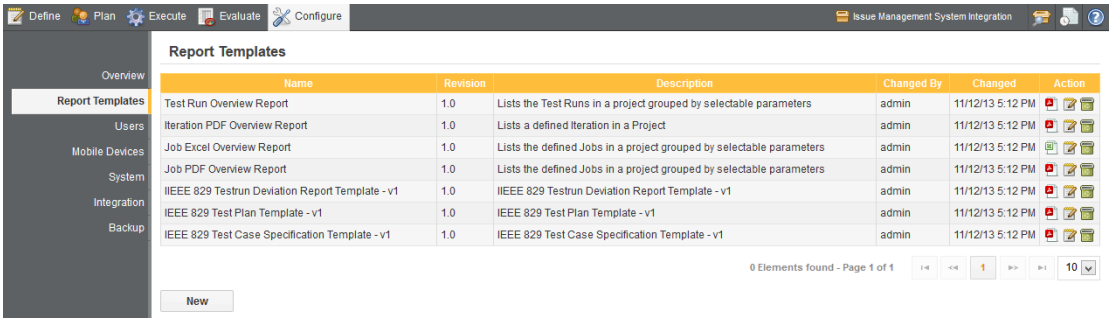

Define
Plan
Execute
Evaluate
Configure
Issue Management System Integration
Overview
Report Templates
Users
Mobile Devices
System
Integration
Backup
Report Templates
Name
Revision
Description
Changed By
Changed
Action
Test Run Overview Report 1.0 Lists the Test Runs in a project grouped by selectable parameters admin 11/12/13 5:12 PM
Iteration PDF Overview Report 1.0 Lists a defined Iteration in a Project admin 11/12/13 5:12 PM
Job Excel Overview Report 1.0 Lists the defined Jobs in a project grouped by selectable parameters admin 11/12/13 5:12 PM
Job PDF Overview Report 1.0 Lists the defined Jobs in a project grouped by selectable parameters admin 11/12/13 5:12 PM
IIEEE 829 Testrun Deviation Report Template - v1 1.0 IIEEE 829 Testrun Deviation Report Template - v1 admin 11/12/13 5:12 PM
IEEE 829 Test Plan Template - v1 1.0 IEEE 829 Test Plan Template - v1 admin 11/12/13 5:12 PM
IEEE 829 Test Case Specification Template - v1 1.0 IEEE 829 Test Case Specification Template - v1 admin 11/12/13 5:12 PM
0 Elements found - Page 1 of 1
New

Figure 10.2. User-Defined Report Template Page

If the user defined report is parameterized, all needed parameters must be entered in order to generate the report.

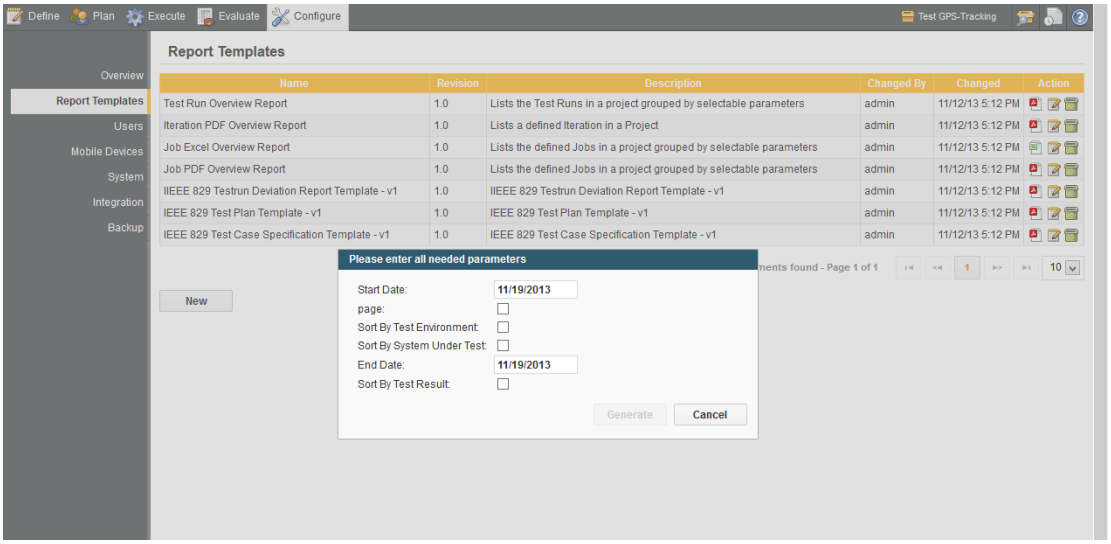


Figure 10.3. Generate a parameterized Report

Note

The **Generate** button becomes visible after entering all required parameters.

To create a new report click the **New** button. A new report is created and the *Report Details* page is shown. To edit an existing report click the icon. Clicking the icon removes the corresponding report from the report repository.

10.2.1. Report Details

On the *Report Details* page it is possible to set the name and description of the report. A valid script and a report template must also be provided in order to save the user-defined report. It is possible to edit the script and template in the text area or upload them directly from an external file.

Clicking the **Save** button causes Klaros-Testmanagement to attempt to save the report to the content repository.

Note

The report is only saved if the script and template are valid and free of errors.

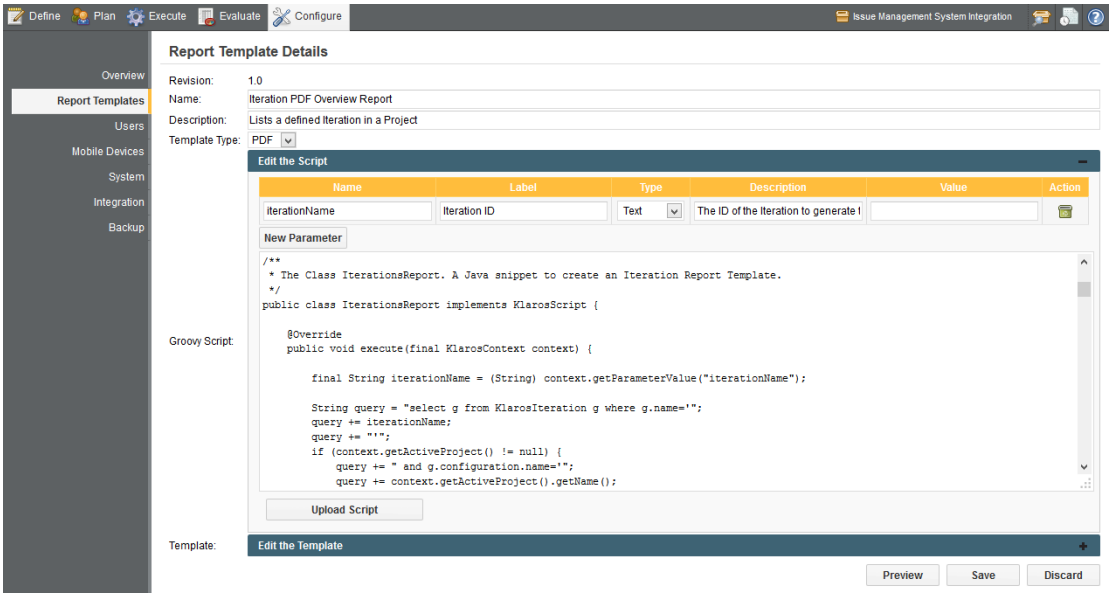


Figure 10.4. Report Details

Further information about creating user-defined reports is available in Section 11.1, “Create A New Report Template”.

10.3. Users

The *Users* section is used to create and edit Klaros-Testmanagement user accounts.

The *Maintain Users* page gives the user an overview of the user accounts in Klaros-Testmanagement. The accounts can be created, enabled, disabled and edited here. (Figure 10.5). Only users with an *Administrator* role can edit all accounts, others are only able to edit their own account settings.

User accounts can be viewed on a printer-friendly page as described in Section 5.2.6, “Print Pages”.

Changing User Roles

Users can not change their own *System Account* flag or *Role*.

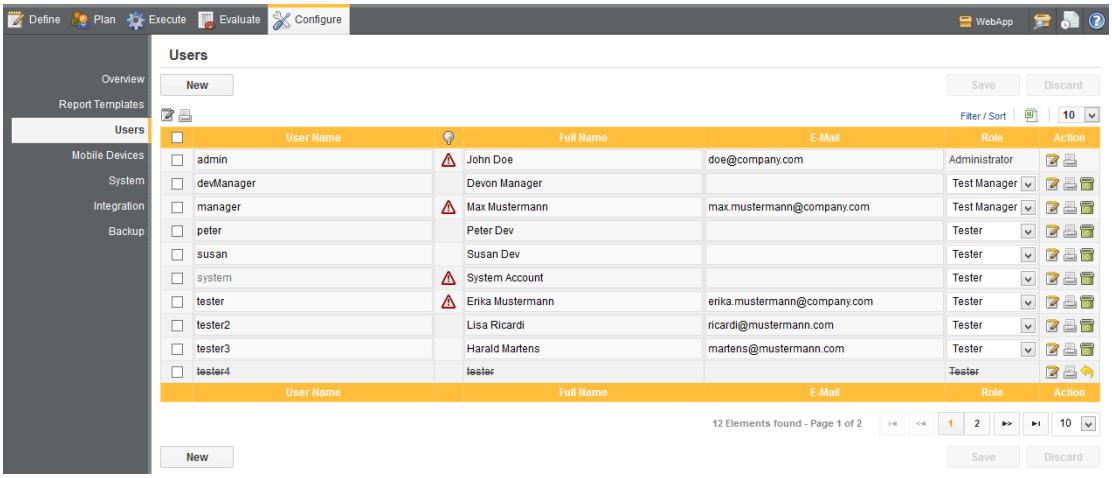


Figure 10.5. The Maintain Users Page

10.3.1. Creating a User

Clicking the **New** button creates a new user account (Figure 10.5). The *User Name*, *Full Name*, *E-Mail Address* and *Role* attributes can be set here directly. When the **Save** button is clicked, a dialog will appear for each new user account to set the *Password* and *System Account* flag (Figure 10.6). Per default users of role *Administrator* or *Manager* are able to create new user accounts. Users of role *Manager* can only create accounts of role *Tester* or *Guest*.

Limit account creation to administrators

If a more stringent approach is desired, the *Only Administrators can create Users* setting can be used to limit user creation to the administrator role. See Section 10.5.1, “Miscellaneous Settings” for more information.

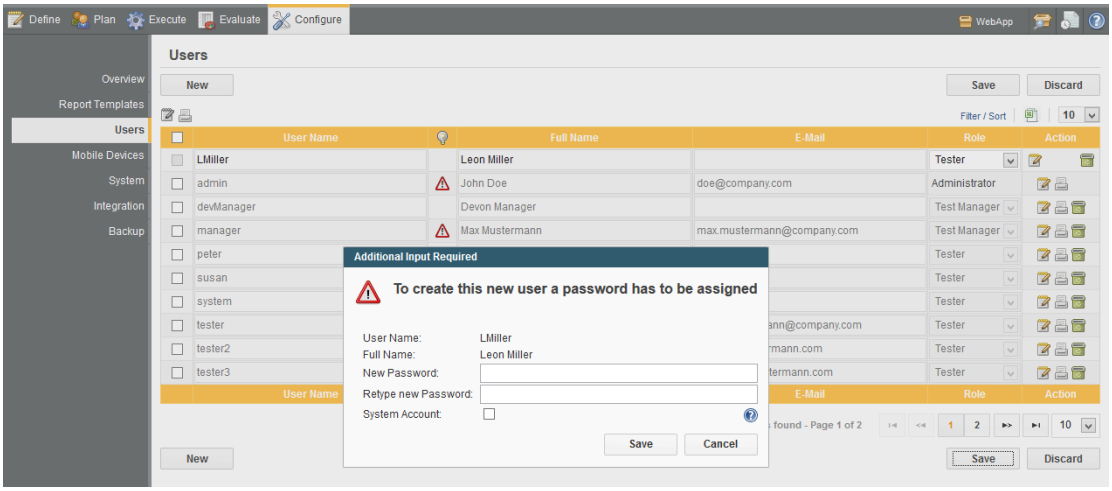


Figure 10.6. The Save New User Dialog

10.3.2. Deleting a User

Clicking the icon on the *Maintain User Page* (Figure 10.5) causes the user account to be disabled. Only users of role *Administrator* can see and restore disabled users.

10.3.3. User Details

This page (Figure 10.7) allows the user to edit the following attributes of a user account:

- *Full Name*

  The full name of the user

- *User Name*

  The login name of the user

- *E-Mail Address*

  The e-mail address of the user, is used for notification emails. If email notifications are configured (see Section 10.3.1, “Creating a User”) and an email address is provided while creating a new user (see Section 10.5.2, “Notification Settings”), this user will receive a registration email. This email contains the new user's username and password.

- *System Account*

  If this flag is set the user is not able to login at the login page and interactively control the application. System accounts should be used for automated tasks like importing data.

- *Password*

  The password to log in to Klaros-Testmanagement

- *Retype Password*

  Confirmation of the password

- *User Role*

  The role of the user, which can be *Administrator*, *Manager*, *Tester* or *Guest*.

Changing User Roles

Users can not change their own *System Account* flag or *Role*.

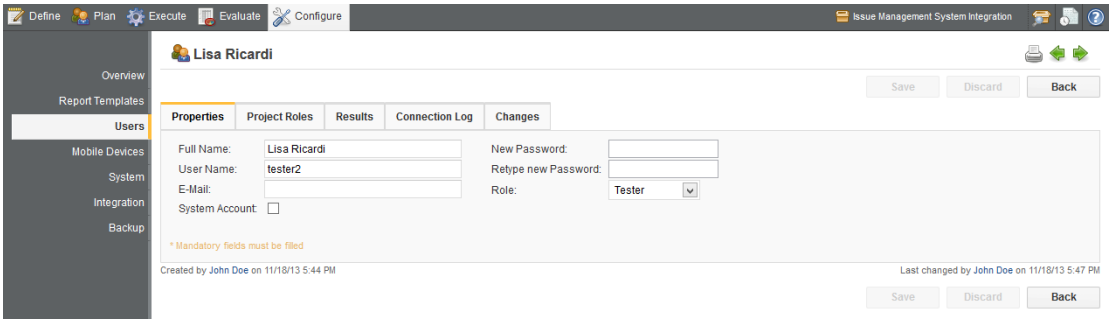


Figure 10.7. The Edit User Screen

Clicking the **Save** button submits the changes while clicking the **Cancel** button discards them.

10.3.3.1. Results

This section contains a *Results* tab as described in Section 5.2.14, “Related Test Runs and Results”.

10.3.3.2. Project Roles

enterprise edition Feature only available in Klaros-Testmanagement Enterprise Edition

The *Project Roles* tab shows the project roles the user has for all project with secured access (see Section 6.1.6.4, “Securing access to the Project” for more information on how to secure access to projects). Administrators can change the project roles for all users and projects here.

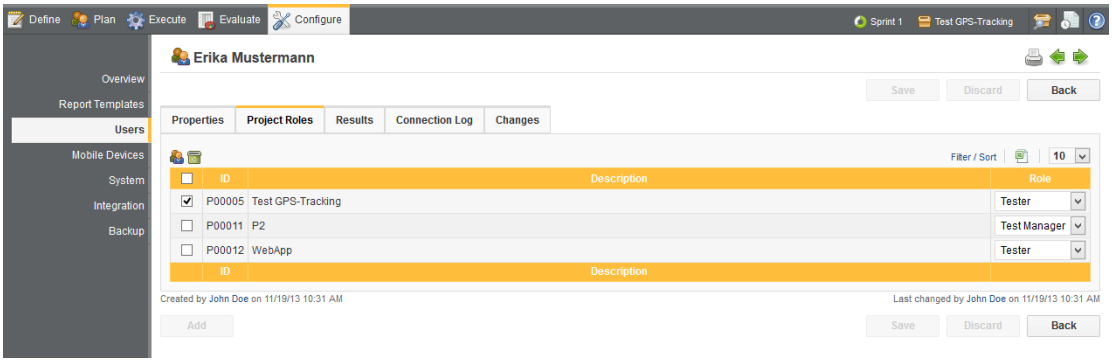


Figure 10.8. The Project Roles Tab

Pressing the **Assign** button opens up a popup window with a list of all projects with project specific roles that the user isn't assigned to yet. Pressing the **Ok** button assigns the user to the selected projects.

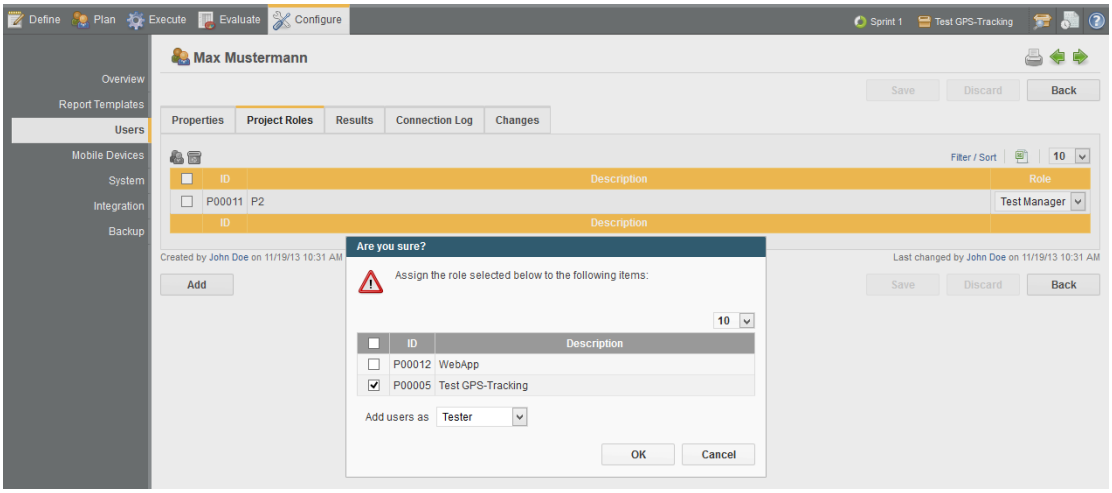


Figure 10.9. The Project Roles Tab

10.3.3.3. Connection Log

mobile edition Only available in Klaros-Testmanagement Mobile Edition.

The *Connection Log* tab displays the connection log for this user. This shows all login attempts of this user from a mobile device with the Klaros-Testmanagement Mobile Edition Android Client installed. The time that each login attempt has happened is also displayed as is the device from which the synchronization login has been attempted. Clicking the device name open up the details page for this mobile device.

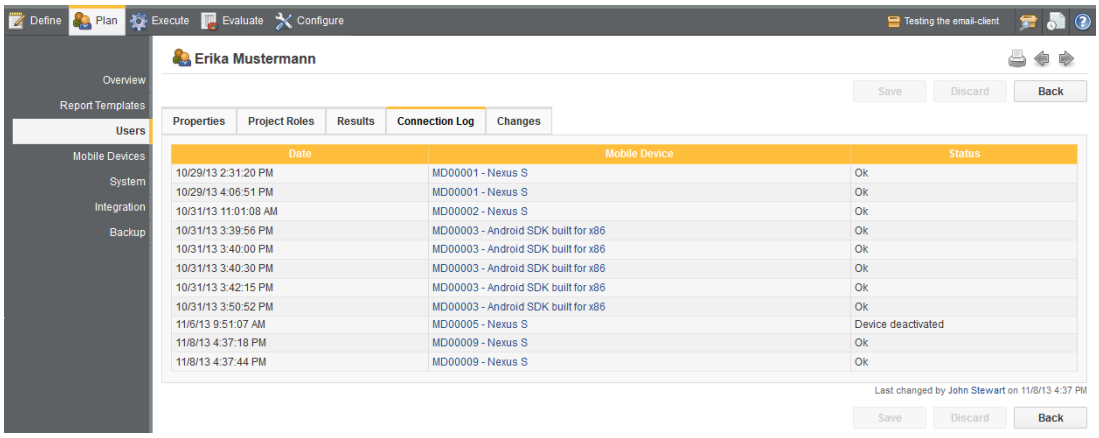


Figure 10.10. The Mobile Device Connection Log Tab

10.4. Mobile Devices

mobile edition Only available in Klaros-Testmanagement Mobile Edition.

The *Mobile Devices* page gives the user an overview of all mobile devices with the Klaros-Testmanagement Mobile Edition Android Client installed that have tried to synchronize with the Klaros-Testmanagement server at least once.

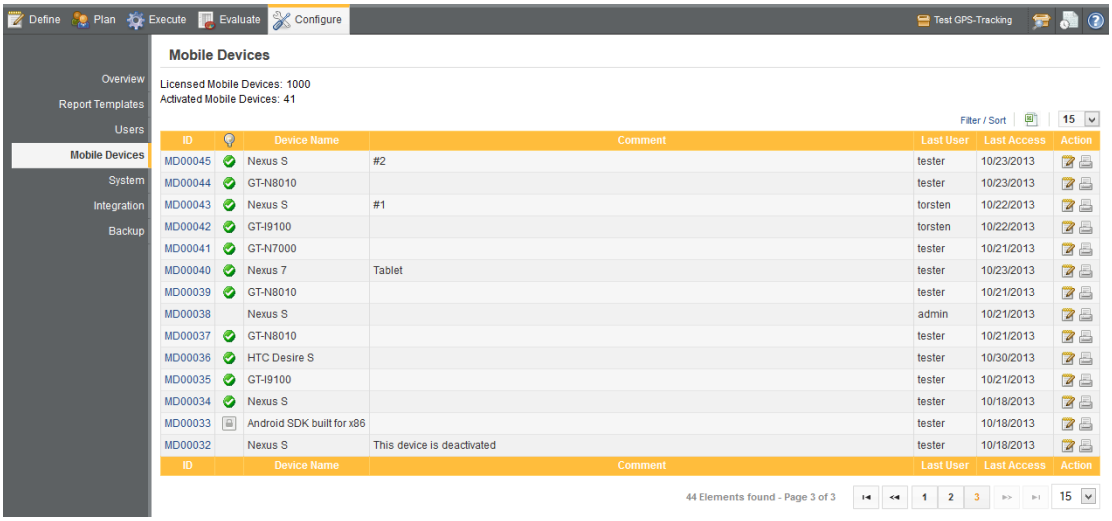


Figure 10.11. The Mobile Devices Page

The first column of the table shows the automatically generated number that each mobile device is assigned. The second info column, indicated by the light bulb, shows the status of the mobile device, which can be either *activated*, *deactivated* or *blocked*.

- Activated devices ( ) can receive test artifacts and also send test results back to Klaros-Testmanagement. If a mobile device connects to Klaros-Testmanagement for the first time, it is automatically activated if enough free mobile device slots are available.

- Deactivated devices ( ) cannot access any resources. When a user tries to connect to Klaros-Testmanagement, he receives an error message. If a mobile device connects to Klaros-Testmanagement for the first time and if there are no free mobile device slots available, then the mobile device is deactivated. It can later be activated if another mobile device gets deactivated first.

- Blocked devices ( ) cannot access any resources. When a user tries to connect to Klaros-Testmanagement, he receives an error message.

Above the table two numbers are displayed, *Licensed Mobile Devices* and *Activated Mobile Devices*. *Licensed Mobile Devices* shows the maximum number of mobile devices that can be activated on this Klaros-Testmanagement installation. *Activated Mobile Devices* shows the number of mobile devices that are currently activated. This number is less or equal to the number of licensed mobile devices.

The column labeled *Device Name* shows the model name of the mobile device. To further distinguish multiple mobile devices of the same model, the *Comment* column can be used. The last two columns show the user name of the last user that has synchronized with this mobile device and also the time at which the synchronization took place. Clicking the username displays the user details page for this user.

Mobile devices may be viewed on a printer-friendly page as described in Section 5.2.6, “Print Pages”.

10.4.1. Mobile Device Details

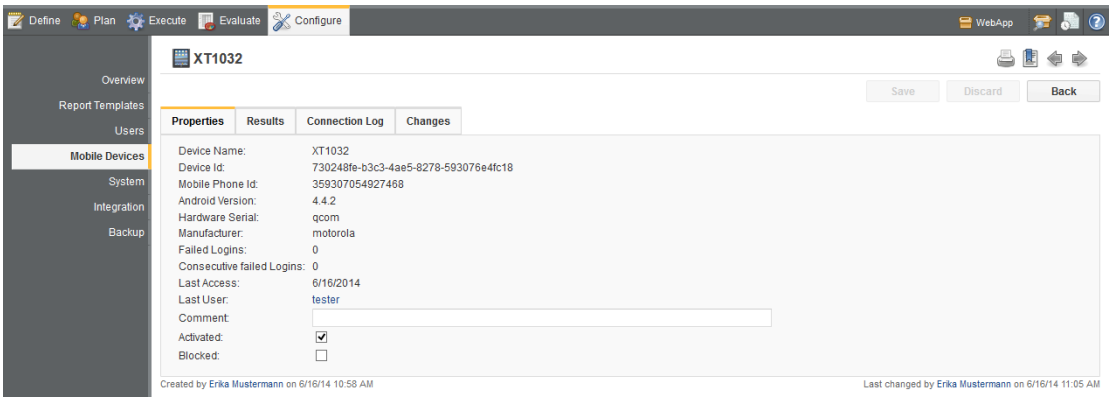


Figure 10.12. The Mobile Devices Details Page

Clicking either the id of the mobile device or the details symbol on the rightmost column opens the mobile device details page. This page contains all of the details of the mobile device, like the installed Android version and the unique device id. An administrator can check the *Activated* checkbox to activate or deactivate the device and the *Blocked* checkbox to block or unblock it.

The *Results* tab displays the results of all test runs that have been executed on this mobile device.

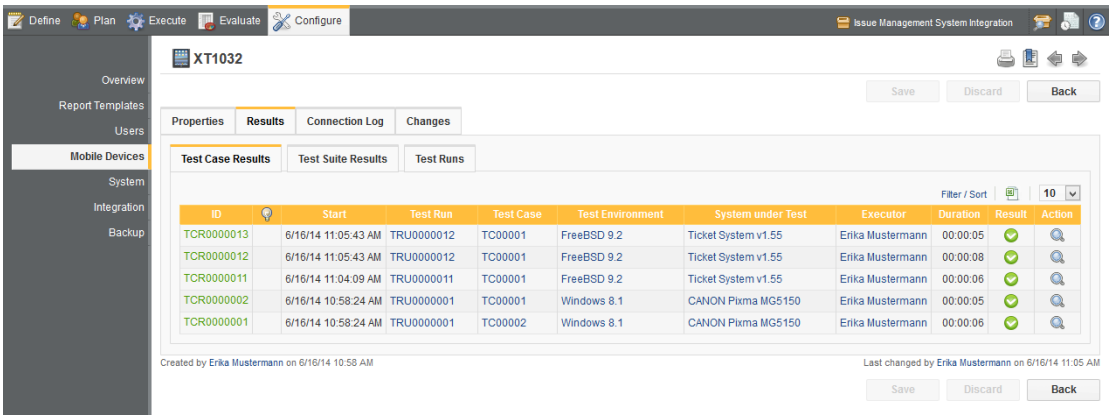


Figure 10.13. The Mobile Device Results Tab

The *Connection Log* tab displays the connection log for the mobile device. It shows the time and user name used for each synchronization of this device with Klaros-Testmanagement. Clicking the user name opens up the user details page for this user.

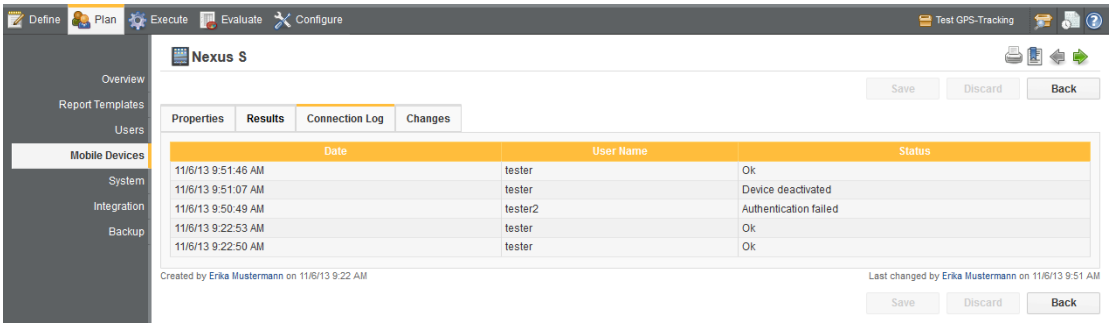


Figure 10.14. The Mobile Device Connection Log Tab

10.5. System

This page contains the settings for the appearance and behavior of Klaros-Testmanagement. It contains four tabs, *Miscellaneous*, *Notifications*, *Interface* and *Languages*.

Clicking the **Save** button persists the changes to the Klaros-Testmanagement properties file, and clicking **Cancel** discards all changes.

10.5.1. Miscellaneous Settings

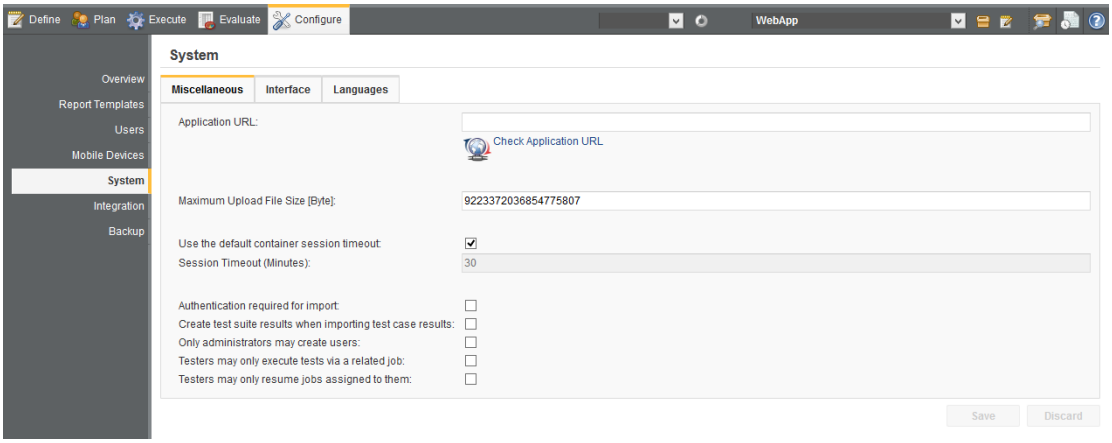


Figure 10.15. The Miscellaneous Settings Tab

This tab contains settings which govern the behavior of Klaros-Testmanagement. The following settings can be controlled here:

- *Application URL*

  This URL must be set if the application runs behind a proxy. The proxy is responsible to redirect the user request to a physical URL. More information about rewriting URLs can be found at http://httpd.apache.org/docs/2.0/mod/mod_proxy.html.

  Press the *Check Application URL* link to test if the URL has been entered correctly.

- *Maximum Upload File Size [Byte]*

  The maximum attachment size which can be uploaded. Changing this value requires a restart of the application server.

- *Use the default container session timeout*

  If checked, the default tomcat session timeout will be used (30 minutes).

- *Session Timeout (Minutes)*

  If the checkbox above is unchecked, this value is used as the session timeout delay.

- *Authentication required for import*

  If true, user credentials are required to import results (e.g. using Jenkins/Hudson plugin). See also Section 12.5.2.1, "Jenkins/Hudson Plugin for Klaros-Testmanagement".

- *Create test suite results when importing test case results*

  If true, test suite results are automatically created for the corresponding test suite information contained in the result file. This information may vary with the import format. Additionally a corresponding test suite is created for the test suite result if it did not yet exist.

- *Only Administrators can create Users*

  If checked, only administrators can create users. If unchecked, managers can also create users.

- *Testers may only execute tests via a related job*

  If checked, users in the role of *Tester* may only execute tests by executing a Job.

- *Testers may only resume jobs assigned to them*

  If checked, users in the role of *Tester* may only resume executions of Jobs which are assigned to them.

10.5.2. Notification Settings

Notifications will be sent by the system via email for various events like creating a user account or assigning a job. This section allows to configure the conditions under which these notification emails will be sent.

Sets of notifications can be defined in a notification scheme and reused for various projects. If a notification scheme is declared as the default notification scheme, it will automatically be assigned to every existing project unless a different scheme is defined for this project.

Figure 10.16 shows the notification scheme definition tab.

Email-Configuration required!

For notifications to work properly it is required to both configure the email server settings as described in Section 10.6.2, "E-Mail" and make sure that all user accounts contain a valid email address.

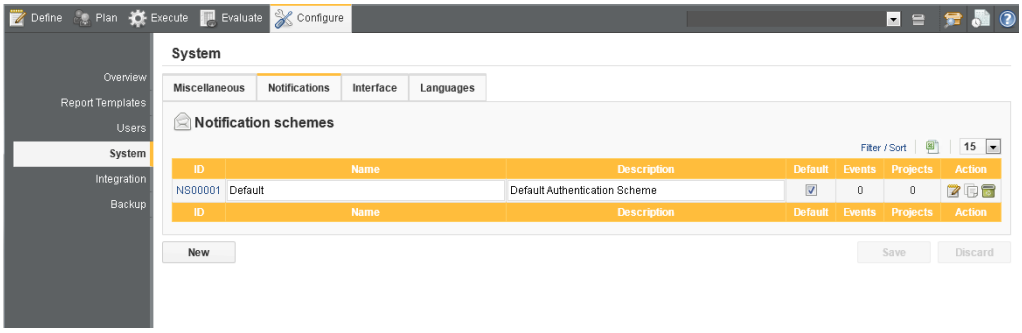


Figure 10.16. The Notification Schemes Tab

Clicking the **New** button creates a notification scheme. The *Name* and *Description* fields can be set here directly. If the *Default* check box is activated this notification scheme will be the default notification scheme which is per default applied to every project.

By pressing the icon the notification edit tab is shown where the notifications and project assignments can be defined. Clicking the icon causes the notification scheme to be scheduled for deletion.

When the **Save** button is clicked the notification scheme will be saved. The **Cancel** discards the changes made.

10.5.2.1. Notification Scheme Settings

Figure 10.17 shows the edit options for a notification scheme.

The list shows the notifications already defined for the notification scheme currently edited. For creating a notification, the user may individually assign events from a predefined set of available notification events to one or more notification targets (user accounts).

The available notification events are:

- *Job Assigned*

  This event is triggered once a Job is saved with a new assignee.

- *Account Created*

  This event is triggered once a user account has been created.

- *Account Password Changed*

  This event is triggered once an account password has been changed in the local user database.

The available notification targets are:

- *Assignee*

  The assigned user account for a job assignment or the currently created/changed user account.

- *Creator*

  The user responsible for triggering the notification event, e.g. by assigning a job.

- *Current User*

  The user account currently active.

- *Specific User*

  A specific user account.

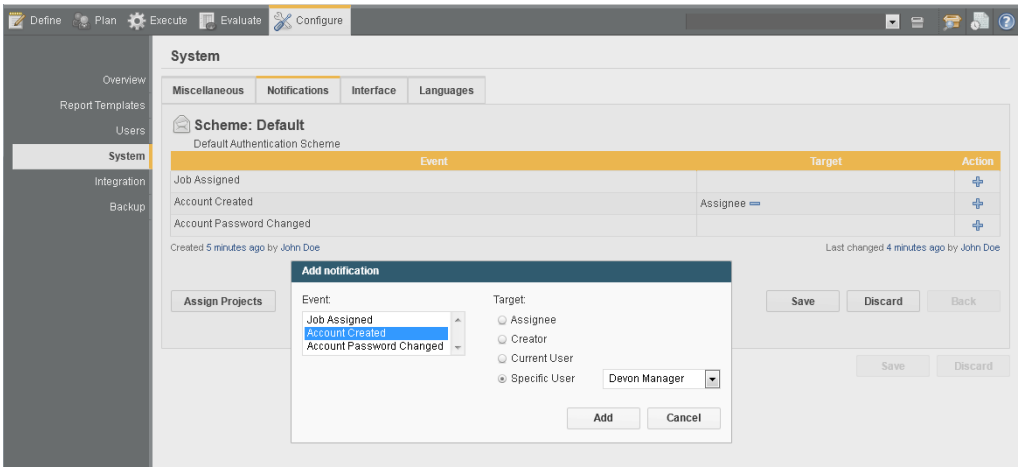


Figure 10.17. The Notifications Tab

Using the **Assign Projects** button the user may assign a notification scheme to a limited set of projects, thus overriding the default notification scheme.

Figure 10.18 shows the dialog for project assignement.

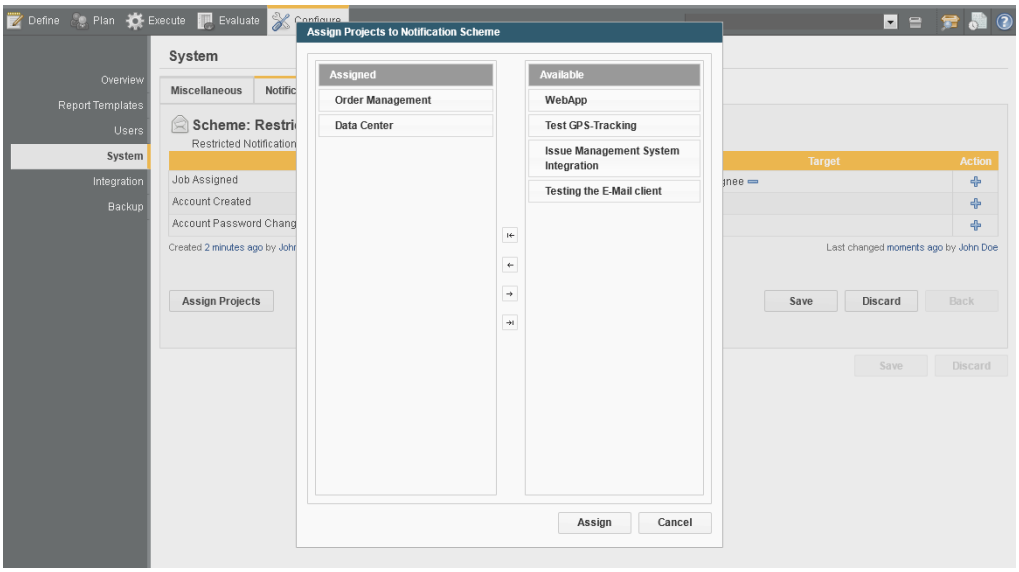


Figure 10.18. Assigning Projects to a Notification Scheme

10.5.3. Interface Settings

This tab contains settings which govern the behavior of the user interface in Klaros-Test-management.

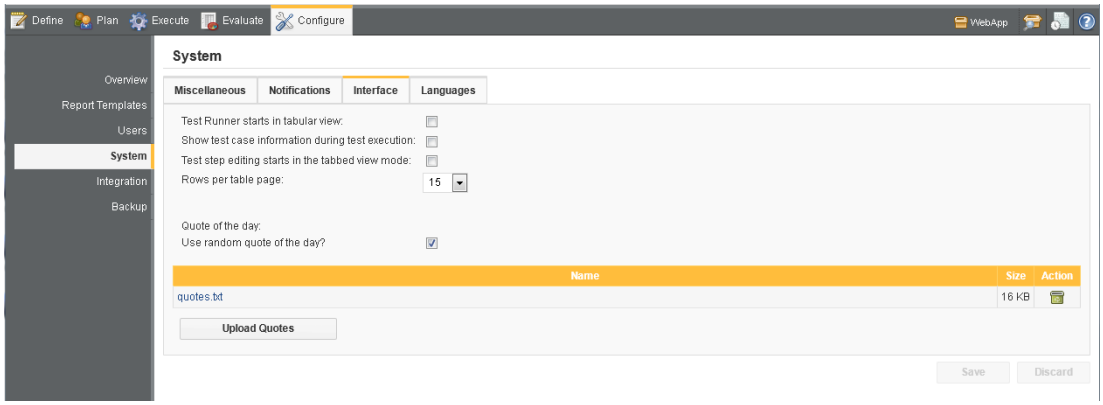


Figure 10.19. The Interface Settings Tab

The following settings can be controlled here:

- *Test Runner starts in tabular view*

  If true the testrunner will per default show the tabluar view.

- *Show test case information during test execution*

  If true, the test case detail information panel is always expanded per default when launching or executing tests.

- *Test step editing starts in the tabbed view mode*

  If true, the default view in the *Steps* tab in the *Test Case Details* section will be the tabular view.

- *Rows per table page*

  With this option, the number of rows per table page can be set.

- *Quote of the day*

  The message to display on the login screen.

- *Use random quote of the day*

  If true, a line from the file uploaded below will be displayed as the quote of the day.

- *Upload quotes*

  This allows the user to upload a quotes file to be used for the random quote of the day. This should be a text file named quotes.txt with one quote per line. The details of the current quotes file are shown below, including a download link for the file itself, and a delete button, which causes the default quotes file to be restored.

Caution

Please refer to Section 4.2, “Custom Quotes” for detailed information on the file format.

10.5.4. Languages

Klaros-Testmanagement supports the addition of custom languages. The available languages can be set in the *Languages* section of the system parameters section. The language settings page is shown in Figure 10.20, “The Languages Tab”.

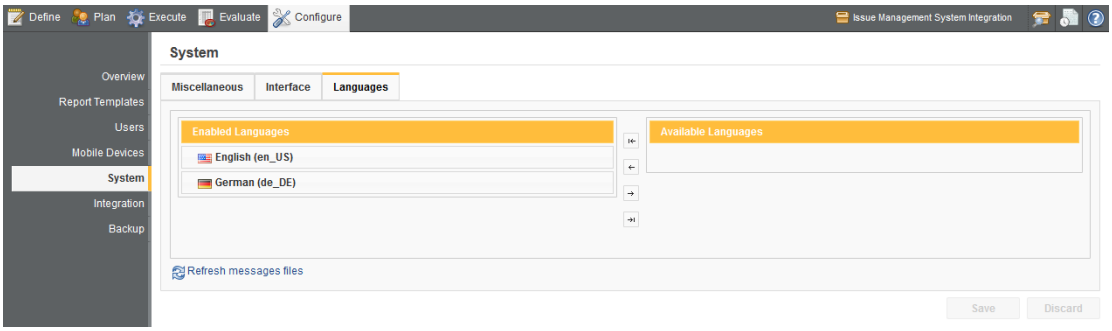


Figure 10.20. The Languages Tab

10.5.4.1. Enabling and Disabling Languages

Up to 5 languages can be enabled at once. Enabled languages are added to the language selection panel at the top of the screen in Klaros. To enable a language, click the icon in the *Available Languages* table and to disable a language, click the icon in the *Enabled Languages* table.

Caution

Please refer to Section 4.1, “Languages” for detailed information on the file format.

10.6. Integration

This page contains the settings for integration of Klaros-Testmanagement with external tools and infrastructure. It contains four tabs, *E-Mail*, *Issue Management*, *Network* and *LDAP*.

Clicking the **Save** button persists the changes to the Klaros-Testmanagement properties file, and clicking **Cancel** discards all changes.

10.6.1. Issue Management

An Issue Management System (aka Issue Tracker, Bug Tracker) is a software package that manages and maintains issues occuring during the testing process. Issues may contain reports about defects in a software system or other observed information.

Klaros-Testmanagement is capable of creating and retrieving entries in remote issue management systems and assigning them to failed test results. It is possible to configure and simultaneously use multiple issue management systems.

Currently Klaros-Testmanagement supports the following issue management systems:

- Bugzilla (a free open source issue management system, see `http://www.bugzilla.org/`)
- JIRA (a commercial issue management system produced by Atlassian Pty Ltd., see `http://www.atlassian.com/`)
- Mantis (a free open source issue management system, see `http://www.mantisbt.org/`)
- Redmine (a free open source issue management system, see `http://www.redmine.org/`)
- Trac (a free open source issue management system, see `http://trac.edgewall.org/`)

Remote System Configuration may be Required

Please refer to Section 3.12, “Configuring External Issue Management Systems” for detailed information on prerequisites for the different issue management systems. Some systems need to be configured before they can be connected to.

These issue management systems are configured in the *Issue Management* section in the Configure menu. The page is shown in Figure 10.21, “The Issue Management Tab”.

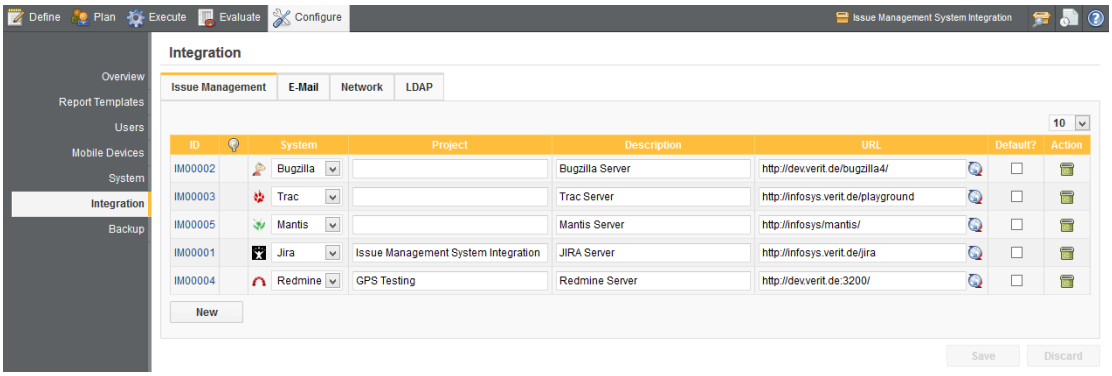


Figure 10.21. The Issue Management Tab

The issue management systems sections shows all of the configured systems in a single table. The properties of each entry can be changed directly in the table.

The properties of the issue management system are:

ID — The internal id of the issue management system. This property is assigned automatically and cannot be altered by the user.

Info — If the deletion of an issue management system is prohibited since there are test case results assigned to issues in the issue management system, the info column shows a icon.

To enable background synchronisation of issues it is required that credentials are configured for the issue management system via the (icon). If credentials are not yet available a icon is shown.

System — The system column indicates which issue management system is used. Currently Klaros-Testmanagement supports JIRA, Trac, Mantis, Redmine and Bugzilla.

Project — If the issue management system organizes Issues in projects, it is possible to specify the project id where the new Issues should be created in the system. JIRA, Redmine, Mantis and Bugzilla manages their issues in different projects. For Trac, the URL is used to specify different projects.

Description — In the *Description* field a description of the issue management system can be entered.

URL — In the URL field the link to the issue management system is specified. To check if the URL is valid, the *Validate the URL* button on the right of the URL field can be pressed. If the URL to the issue management system is configured correctly, a confirmation message will be displayed in the message area.

Trailing slashes in URLs

In case of an authentication error, first check whether the addition or deletion of a trailing slash resolves the issue.

Action — The action column contains a button that can be used to delete an system from the configuration. If the issue management system is unused, i.e. it is not used in any project, the configuration of the system can be deleted.

Project selection

The valid project ID values can be found in the issue management systems at the following locations.

Bugzilla Project ID — The Bugzilla Project ID consists of the `Product` field value as shown below.

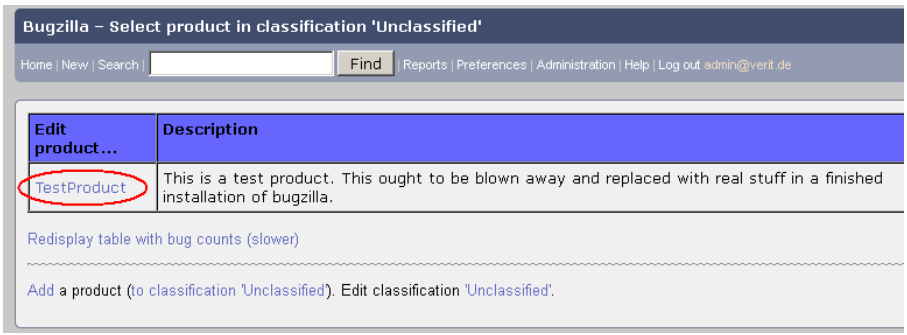

Bugzilla – Select product in classification 'Unclassified'
Home | New | Search |
Find
| Reports | Preferences | Administration | Help | Log out admin@verit.de
Edit product...
Description
TestProduct
This is a test product. This ought to be blown away and replaced with real stuff in a finished installation of bugzilla.
Redisplay table with bug counts (slower)
Add a product (to classification 'Unclassified'). Edit classification 'Unclassified'.

Figure 10.22. The Bugzilla Project ID

JIRA Project ID — The JIRA Project ID consists of the `Key` field value as shown below.

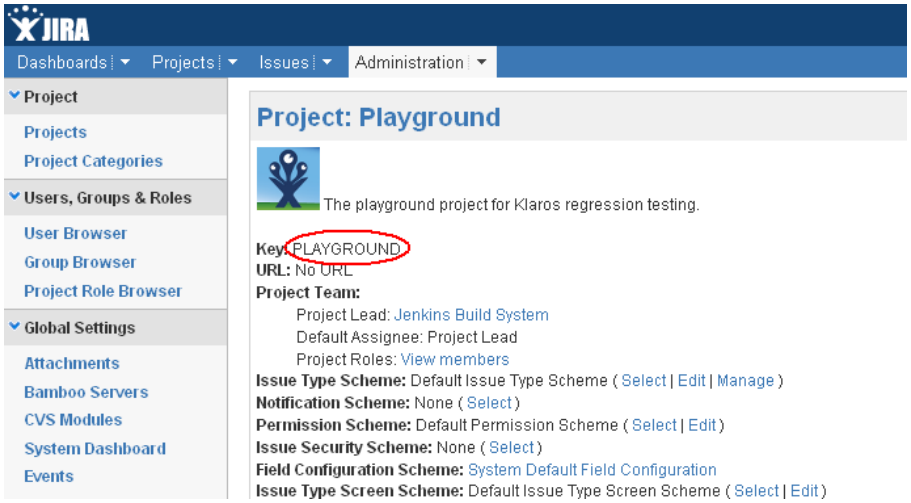


Figure 10.23. The JIRA Project ID

Mantis Project ID — The Mantis Project ID consists of the `Project Name` field value as shown below.

MANTIS
Logged in as: admin (Mantis Administrator - administrator)
2011-05-27 10:20 CEST
Main | My View | View Issues | Report Issue | Change Log | Roadmap | Summary | Docs |
[Manage Users] [Manage Projects] [Manage Custom Fields] [Man
Edit Project
Project Name
Playground
Status
development

Figure 10.24. The Mantis Project ID

Redmine Project ID — The Redmine Project ID consists of the `Identifier` field value as shown below.

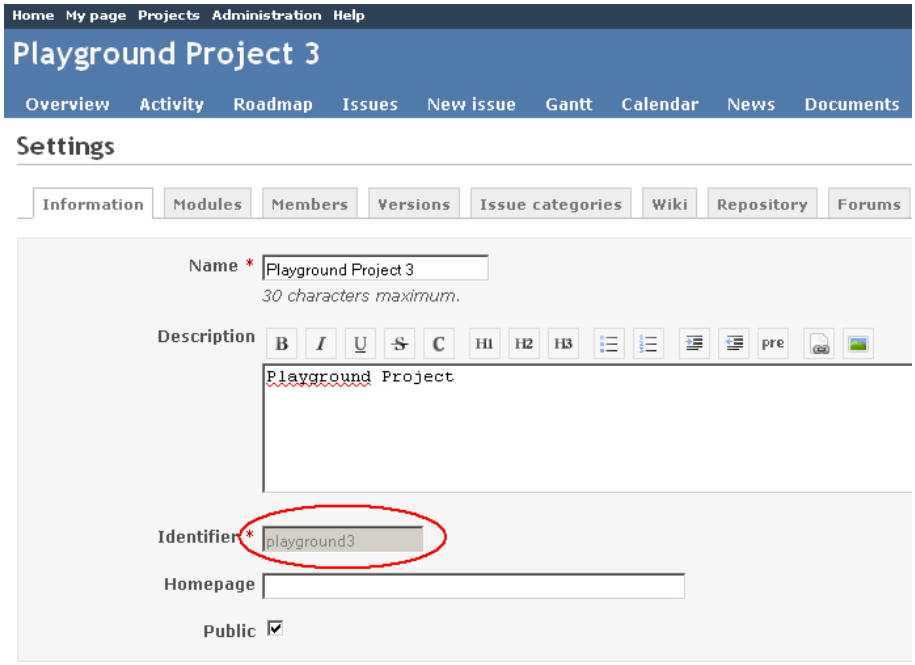


Figure 10.25. The Redmine Project ID

10.6.1.1. Adding a new Issue Management System

To add a new issue management system, click the **New** button. An empty row will be added to the list of issue management systems.

Clicking the **Save** button submits the changes while clicking the **Cancel** button discards them.

10.6.1.2. Editing an existing Issue Management System

The configuration of an issue management system can be changed by editing the fields in the table.

Clicking the **Save** button submits the changes while clicking the **Cancel** button discards them.

10.6.1.3. Deleting an Issue Management System

If the issue management system is not used in any project, it can be deleted by clicking the delete button ( ) of the entry in the table for configuration of the issue management systems.

10.6.2. E-Mail

In the *E-Mail* tab it is possible to change the e-mail server settings of the application. The E-Mail server settings are required for sending notification emails configured in Section 10.5.2, "Notification Settings". The attributes which can be edited are shown in Figure 10.26.

Note

Please consult your system administrator about the values to enter here.

SMTP Server — The host name of your mail server.

SMTP Server Port — The port where your mail server is listening. This value usually depends on the security setting chosen below. The default value is 25 for no security.

Sender Address — The *From* address field of the mails generated by the application.

User Name — A user name needed to authenticate against the SMTP server. This field is only present if SMTP or POP authentication is selected in the *Authentication* field below.

Password — A password needed to authenticate against the SMTP server. This field is only present if SMTP or POP authentication is selected in the *Authentication* field below.

Authentication — Some mail servers require users to authenticate themselves before they allow them to send mails to prevent or identify spammers. This option chooses whether no authentication is required, or a username / password combination via SMTP or POP is used. If one of the latter two is selected the *User Name* and *Password* fields are shown in the user interface.

Security — This setting defines the transport layer security used to access the SMTP mail server. Each transport security setting may require a different port value above. The options are None (Port 25), Secure Socket Layer / SSL (Port 465) or Transport Layer Security / TLS aka STARTTLS (Port 25). Your local port settings may vary from the given defaults.

Send Test Mail — This link sends a test email to verify that your mail server settings are valid. Therefore it is required that your user account contains a valid email address.

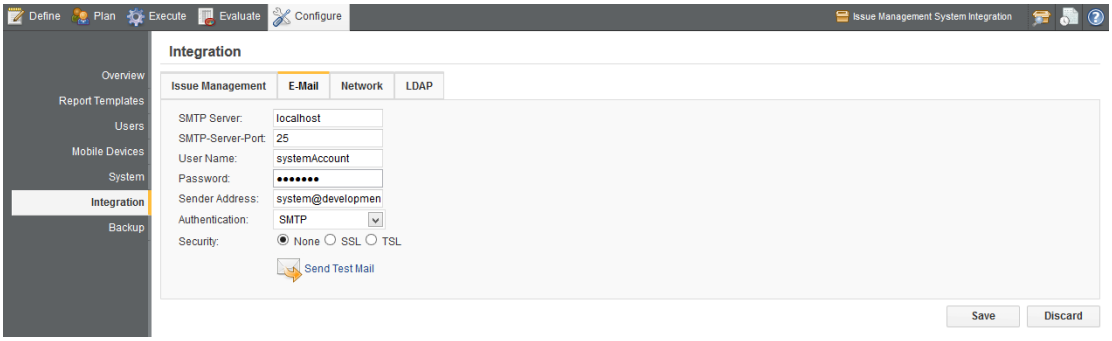


Figure 10.26. The E-Mail Tab

Click the *Send Test Email* link to test if the fields are filled in properly.

10.6.3. Network

Klaros-Testmanagement supports the use of HTTP and SOCKS proxies for network connections. Proxy settings are configured in the *Network* section of the configure menu. The network page is shown in Figure 10.27, “The Network Tab”.

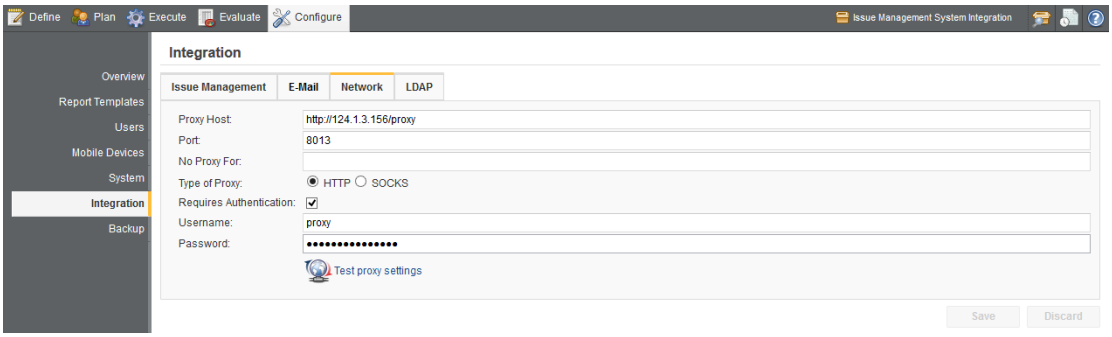


Figure 10.27. The Network Tab

Application URL must be set

If Klaros-Testmanagement is run behind a proxy, the Application URL must be set in order to view some images within Klaros-Testmanagement properly. The Application URL can be set in the *Miscellaneous* tab of *General Settings* (see Section 10.5.1, “Miscellaneous Settings”).

The properties of a proxy are:

Proxy Host	The network address or hostname of the proxy server.
Port	The port that the proxy server is active on.
No Proxy for	A |-separated list of domains which should bypass the proxy settings.
Type of Proxy	Whether the proxy is a HTTP proxy or a SOCKS proxy.
Requires Authentication	Whether or not the proxy requires authentication.
Username	The username to use for username/password authentication.
Password	The password to use for username/password authentication.

In addition, proxy settings can be tested by clicking the icon.

10.6.4. LDAP

Feature only available in Klaros-Testmanagement Enterprise Edition

Note

In order to access an LDAP or Active Directory server, a rather large set of configuration parameters is required. Your system administrator should be able to help you by providing the correct values.

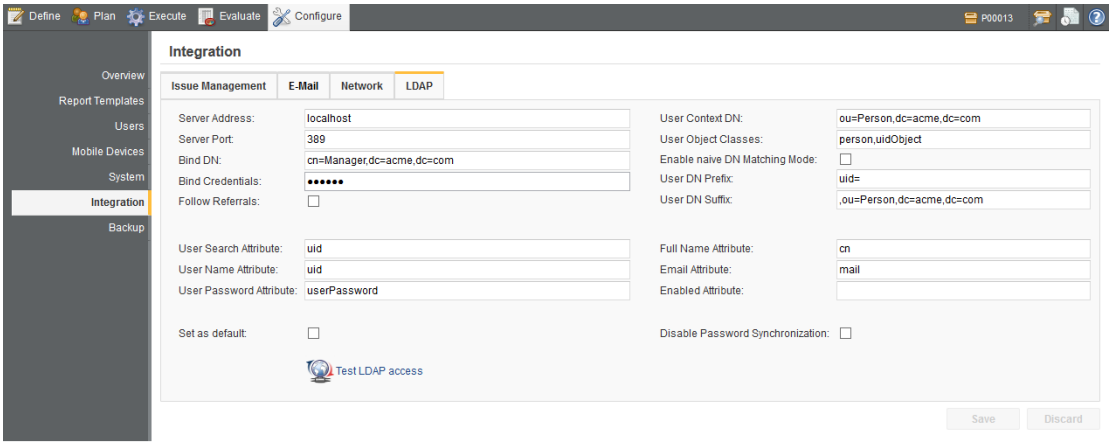


Figure 10.28. The LDAP Tab

Parameters needed to contact the LDAP server:

- *Server Address*

  The URI under which the LDAP server resides (e.g. `ldap.acme.com`).

- *Server Port*

  The port on which the LDAP server is listening (typically `389`).

- *Bind DN*

  The distinguished name used for binding to this LDAP server.

- *Bind Credentials*

  The credentials (password) required to be able to bind to this LDAP server.

- *Follow Referrals*

  If this option is enabled, searching a directory automatically follows any referrals the server might return. When disabled referrals will be ignored and other servers will not be contacted during the search.

Parameters needed to locate user accounts:

- *User Context DN*

  The distinguished name under which user accounts will be searched (e.g. `ou=Users,dc=verit,dc=de`).

- *User Object Classes*

A comma separated list of the LDAP object classes a user account must match to be included in the search (e.g. `person,posixAccount`).

- *Enable naive DN Matching Mode*

  If this option is enabled, a returned DN of a user search is used to identify the user to be authenticated. If it is disabled, the following two parameters will be used in conjunction with the *User Name Attribute* to build a DN to authenticate with.

- *User DN Prefix*

  The distinguished name prefix used to locate user accounts (e.g. `uid=`).

- *User DN Suffix*

  The distinguished name suffix used to locate user accounts (e.g. `,ou=Users,dc=acme,dc=com`). When locating user accounts, the prefix, the account id and the suffix are concatenated to form the distinguished name of the user account.

Parameters describing the attributes of a user account:

- *User Search Attribute*

  The LDAP user name attribute which corresponds to the Klaros-Testmanagement account name (e.g. `uid`).

- *User Name Attribute*

  The LDAP attribute which will be used in the DN bind action which authenticates the user (if no naive DN Matching mode is active). In simple scenarios this will match the *User Search Attribute*. If your LDAP Server setup does not allow you to bind a user with the specified user search attribute you should specify the corresponding attribute here (e.g. `cn`) and use this in conjunction with the corresponding user DN prefix/suffix.

- *User Password Attribute*

  The LDAP password attribute which corresponds to the Klaros-Testmanagement account password (e.g. `userPassword`).

- *Full Name Attribute*

  The LDAP attribute containing the full name of the user account (e.g `cn`). If specified, this will be automatically be transferred into the Klaros-Testmanagement database upon first successful login.

- *Email Attribute*

  The LDAP attribute containing the email address of the user account (e.g. `mail`). If specified, this will automatically be transferred into the Klaros-Testmanagement database upon the first successful login.

- *Enabled Attribute*

  If specified this boolean LDAP attribute decides whether a user will be able to be authenticated. Not all directory servers provide such an attribute.

If the *Set as default* checkmark is activated, the login screen will default to LDAP authentication for all users. It is still possible for existing users to authenticate against the Klaros-Testmanagement user database if selected in the login screen.

Upon the first successful login a matching password hash is created in the local user database so that users can also authenticate themselves against the local user database with their LDAP password. If the *Disable Password Synchronization* checkmark is activated, this synchronization will not be performed and users will only be able to login locally when an administrator assigns them a local password interactively.

Pressing the *Test LDAP access* link tests whether the parameters entered on this page are correct. The test process in divided in two phases.

In the first phase a bind to the server is attempted and a search for all matching users is conducted. If it is successful, a dialog is shown listing the matching users found in the LDAP directory.

In the second phase a username and a password may be entered to test the actual LDAP authentication. The result of the authentication attempt is logged in the logpanel.

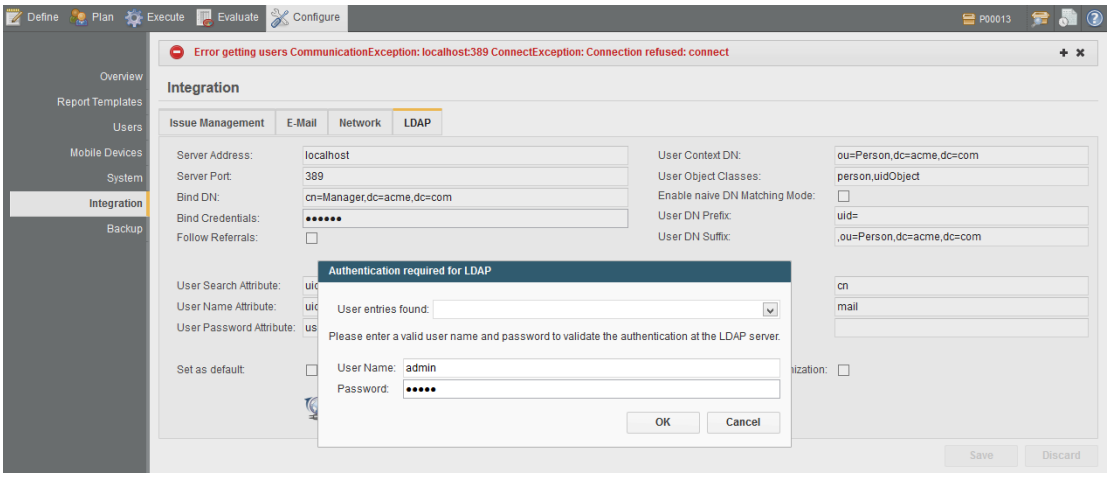


Figure 10.29. The LDAP Authentication Popup

10.7. Backup

To move data between different database installations or to selectively import data, Klaros-Testmanagement provides the functionality to import and export database content via XML files. Only Administrators and Managers are able to import or export projects.

10.7.1. Export

Here it is possible to export selected project data. The user can select one or more projects, or click at the *Select all* option to select all the projects that are available (Figure 10.30).

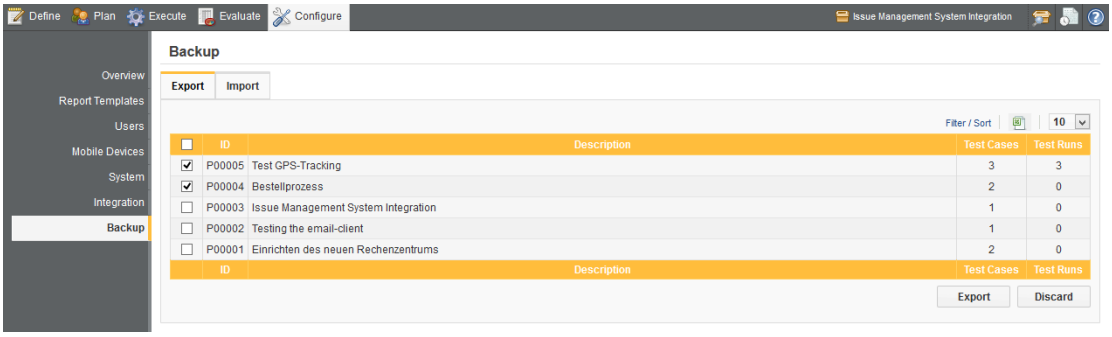


Figure 10.30. The Export Tab

Clicking the **Export** button exports the selected projects to an XML formatted output file and clicking the **Cancel** button discards the selection.

Note

Exported backup files carry a version number identifying the version of the database that was exported. Backup files may only be imported when the database version of the Klaros installation matches the version of the backup file.

10.7.2. Import

Here it is possible to import the project data that has previously been exported. The user can browse for a backup file and upload it. The screen will show the projects contained in the uploaded file, and the user is able to select one or more projects or click the *Select all* option to select all the projects for restoration (Figure 10.31).

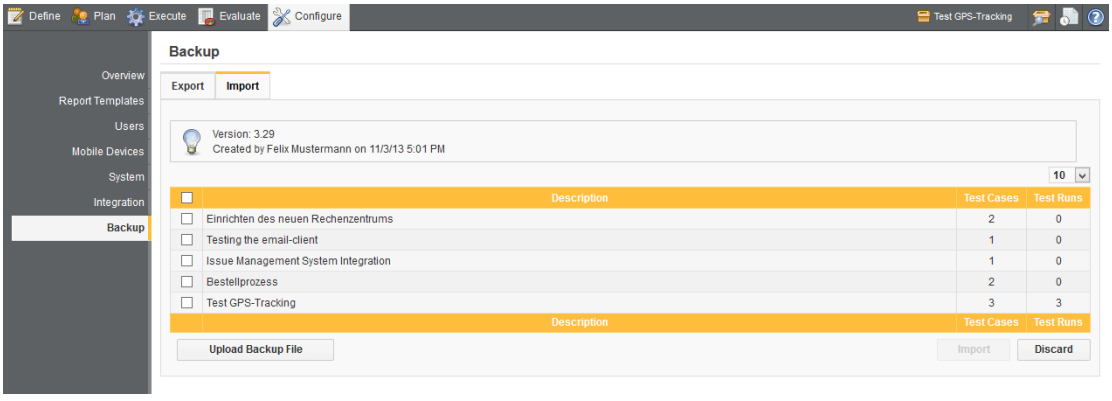


Figure 10.31. The Import Tab

Clicking the **Import** button imports the selected project data and clicking the **Cancel** button cancels the restore.

Note

The import action will not overwrite existing projects or other existing artifacts therein.

Important

If you try to import a backup file from a previous Klaros database version this import will be rejected with a corresponding error message. You need to import that backup in a compatible Klaros release installation first and update that to your current release.

A subsequent export will then yield a compatible backup file.

Chapter 11. Custom Reports

The following chapter covers user defined reports. See Section 9.1.1, "The Overview Reports" on how to generate the pre-configured overview reports that are included in Klaros-Testmanagement Community Edition.

Feature only available in Klaros-Testmanagement Enterprise Edition

With Klaros-Testmanagement Enterprise Edition it is possible to define custom reports. Though Klaros-Testmanagement already provides several reports, it might be helpful to design new reports that suite your and your customer's needs. Figure 11.1, "The Report Generation Process" gives an overview of the report generation process. As the report definition process is based on a Groovy or Java script and SeamPDF or SeamExcel, basic knowledge in Groovy or Java programming and XML conventions are helpful for a report designer.

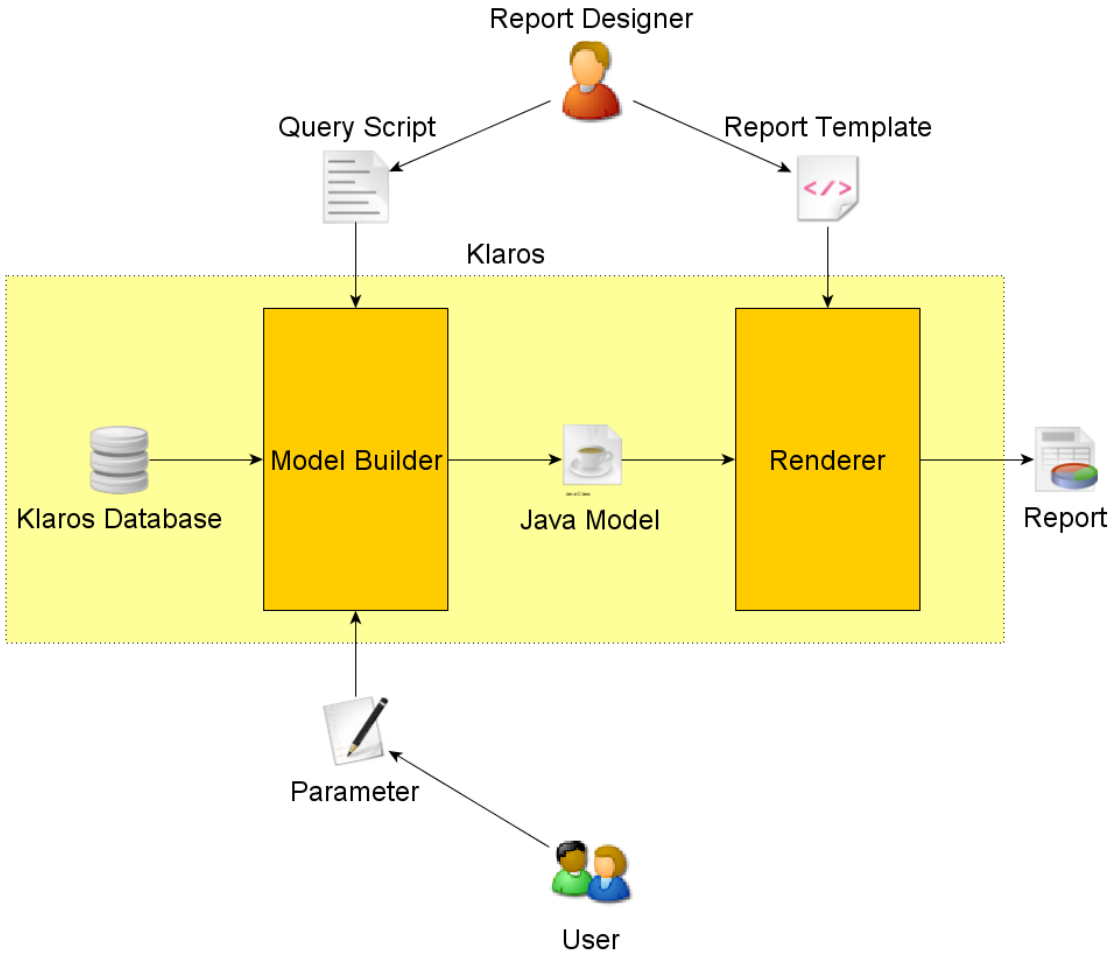


Figure 11.1. The Report Generation Process

The reporting process involves two roles, the `Report Designer` and the `Report User`. While the report designer provides the structure of the report and prepares the report layout, the report user applies the reports to the data collected in Klaros-Testmanagement. The report

designer has to provide a script and a template to build a report. The query script prepares the data and is implemented in Groovy or Java, while the report template describes the layout of the report. The Groovy/Java script taking care of data retrieval and preparation is provided by a class that implements the Section E.2, "KlarosScript Interface". This interface defines a single method called `execute` and takes a de.verit.klaros.scripting.KlarosContext object as input parameter.

To retrieve the data from Klaros-Testmanagement, the report designer can access the de.verit.klaros.core.model via the HQL query language. The main task of the class is to provide and prepare the data for the report. The data for the report template must then be stored in the de.verit.klaros.scripting.KlarosContext object passed to the execute method. To make reports more flexible for report users, it is possible to pass parameters to the Groovy/Java script. The parameters are stored in the context and can be accessed from the script.

Predefined objects in the context

The de.verit.klaros.scripting.KlarosContext object already contains predefined objects. For a list please refer to Section E.1, "Context Variables".

The report template must be implemented using SeamPDF. Section Section 11.3, "Example Report" provides an example of how to define a custom report.

11.1. Create A New Report Template

Before reports can be applied to the test data, they have to be defined. To get to the *Report Templates* page click on the *Configure* icon and select *Report Templates* from the menu on the left side of the screen.

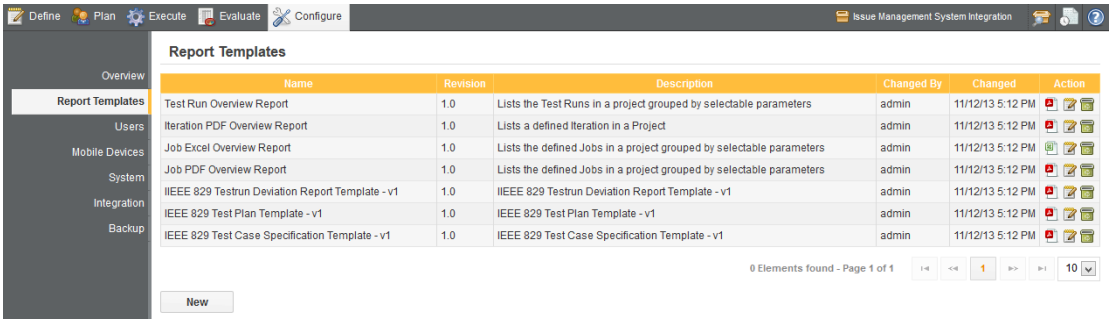

Define
Plan
Execute
Evaluate
Configure
Issue Management System Integration
Overview
Report Templates
Users
Mobile Devices
System
Integration
Backup
Report Templates
Name
Revision
Description
Changed By
Changed
Action
Test Run Overview Report 1.0 Lists the Test Runs in a project grouped by selectable parameters admin 11/12/13 5:12 PM
Iteration PDF Overview Report 1.0 Lists a defined Iteration in a Project admin 11/12/13 5:12 PM
Job Excel Overview Report 1.0 Lists the defined Jobs in a project grouped by selectable parameters admin 11/12/13 5:12 PM
Job PDF Overview Report 1.0 Lists the defined Jobs in a project grouped by selectable parameters admin 11/12/13 5:12 PM
IIEEE 829 Testrun Deviation Report Template - v1 1.0 IIEEE 829 Testrun Deviation Report Template - v1 admin 11/12/13 5:12 PM
IEEE 829 Test Plan Template - v1 1.0 IEEE 829 Test Plan Template - v1 admin 11/12/13 5:12 PM
IEEE 829 Test Case Specification Template - v1 1.0 IEEE 829 Test Case Specification Template - v1 admin 11/12/13 5:12 PM
0 Elements found - Page 1 of 1
1
10
New

Figure 11.2. The Report Templates Page

To create a new report click on the *New* button on the *Report Templates* page. Then provide the basic data for the report by entering a name in the *Name* field and a short description in the *Description* field.

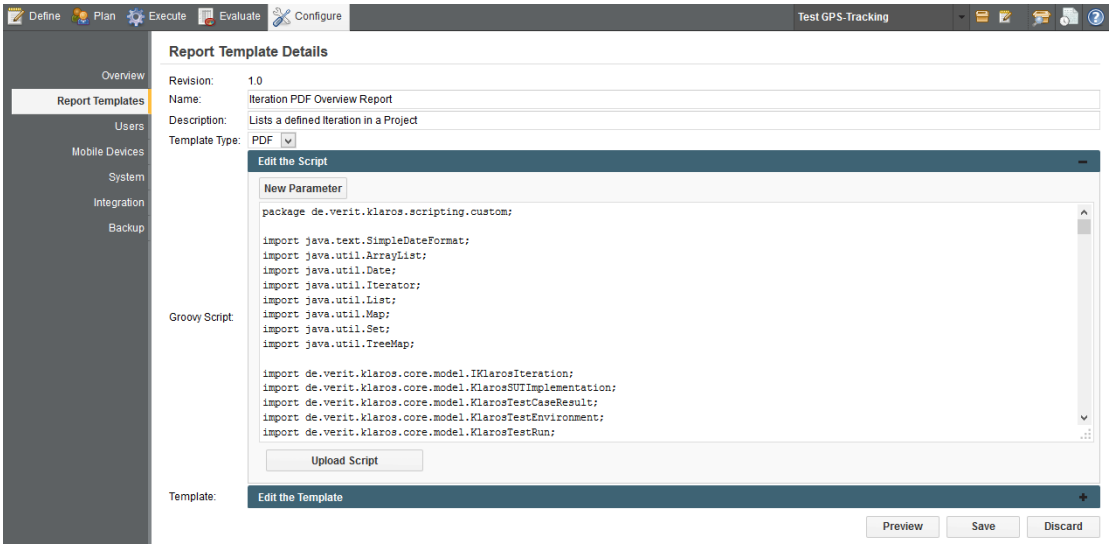


Figure 11.3. The New Report Templates Page

To enter the Groovy script, which retrieves the data, use the *Groovy Script* text area. To unfold the template text area click on the *Edit the Template* link next to the *Template* label. The report's template code can then be entered into the unfolding text area. To unfold the Groovy script text area again click on the *Edit the script* link.

Note

For larger report templates it might be helpful to use a Java IDE e.g. Eclipse, to develop the Groovy script and an XML editor to provide the report template. The created files can then be uploaded into Klaros-Testmanagement. To avoid errors in the Groovy script, add the Klaros model libraries to the build path of your Eclipse project.

Instead of manually entering the code for each text area, the code can be imported from a file. Specify the file to be use by clicking on the **Browse** Button and select the file from the file system. Click *OK* in the file dialog and afterwards click **Upload** to import the selected file into the text area.

This page provides three actions to be executed. To test the code click on the **Preview** button to generate a report. The **Save** button stores the report into the Klaros repository in the file system. The changes can be discarded by pressing the **Back** button.

Note

The generated report is opened automatically in a new browser window. If this does not work for you, please check if you have a pop-up blocker active.

To provide a certain degree of freedom to the report users and to make the reports more flexible, it is possible to pass parameters to the Groovy script. This mechanism can, for example, be used to pass a time span to the Groovy script, so that only data for this time span is retrieved from Klaros-Testmanagement.

To pass arguments to the report click the **New Parameter** button.

A new section as shown in Figure 11.4, "Specifying Parameters" is displayed, which enables the addition of parameters. Clicking the button allows a parameter to be added to the parameter list.

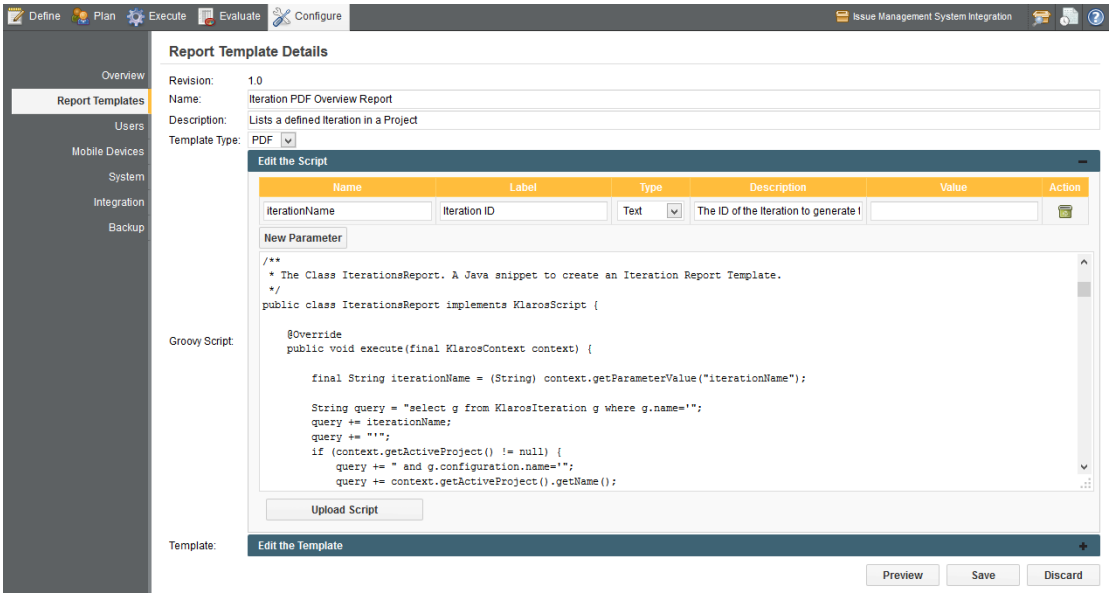


Figure 11.4. Specifying Parameters

Clicking the icon removes the parameter from the list. The type of the parameters can be specified by selecting the appropriate value from combo box. Supported types are `Text`, `Number`, `Date`, `Boolean` `Enumeration` and `Multi-enumeration`.

The passed parameters can be accessed by the Groovy script by either calling the `getParameterValue(String name)` or the `getParameter(String name)` method. The methods will return null if no parameter with the specified name can be found.

11.1.1. Supported Parameter Types

The following parameter types are supported:

Report Parameter Types

- Text
- Number
- Date
- Boolean
- Enumeration
- Multi-enumeration

11.1.2. Dealing with Parameters

As mentioned before the passed parameters can be retrieved from the context by the methods `getParameterValue(String name)` and `getParameter(String name)`.

The following code snippet shows how to access the parameters and how to use them in the Groovy script:

```
query.append("select tcr from KlarosTestCaseResult tcr where tcr.executionTime <");
long executionTime = context.getParameterValue("executionTime");
query.append(executionTime);
List<?> tcr = context.executeQuery(query.toString());
```

This query will retrieve all `KlarosTestCaseResult` instances that have an execution time smaller than the value passed with the parameter `executionTime`.

As an alternative the parameters can be directly accessed in the query as shown in the followingcode snippet.

```
query.append("select tcr from KlarosTestCaseResult tcr where tcr.executionTime <:executionTime");
List<?> tcr = context.executeParameterizedQuery(query.toString());
```

Parameters can be accessed from the report template in the following way:

```
<p:text value="Testresults for testruns with execution time &lt; #{executionTime} ms" />
```

Note

Make sure to escape characters like &, < or > when using them in XML attributes in the report template.

11.2. Applying a Report Template

To apply a report to your test data click on the *Evaluate* icon and select *Report Templates* from the menu on the left side. A list of available report templates will be provided for selection.

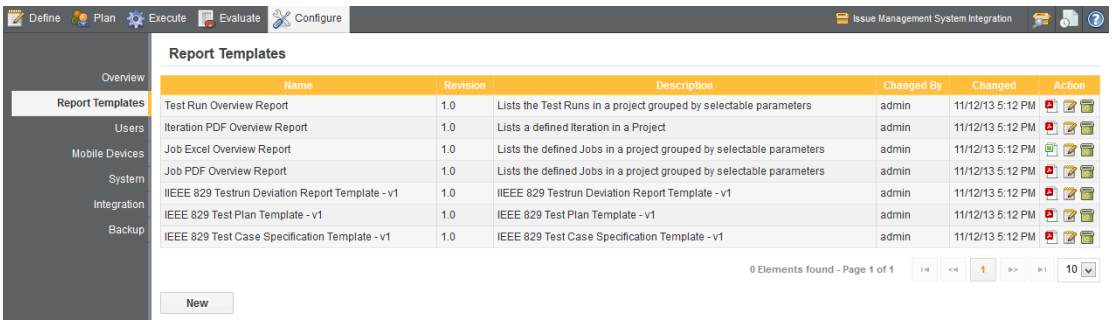


Figure 11.5. Apply a Report Template

Clicking on the or icon of a report template causes a PDF or Excel report to be rendered. If the Groovy script was defined with parameters, a pop-up window will prompt the user to enter the defined parameters before the Report is generated.

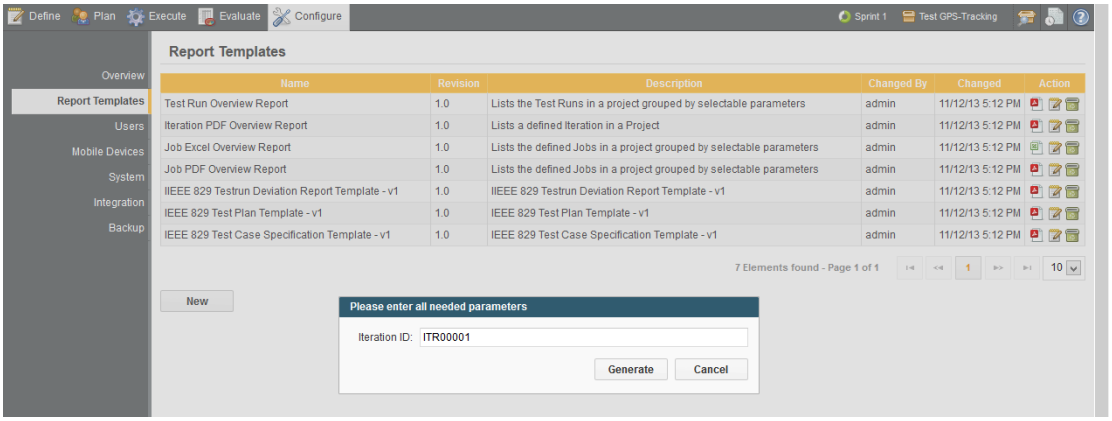


Figure 11.6. Enter Parameter

11.3. Example Report

This section provides an example report, which shows how to retrieve test case results and how to display them depending on their status.

11.3.1. Creating the Groovy Script

The following code snippet shows the frame for a Groovy script with the required imports. The code to retrieve the data must be implemented in the execute method. A more detailed description of the Klaros-Testmanagement API can be found in de.verit.klaros.core.model.

```
import de.verit.klaros.scripting.*;
import de.verit.klaros.core.model.*;
import java.util.*;

public class TestScript implements KlarosScript {

  public void execute(KlarosContext context) {...
  }
}
```

The next step in the data retrieval process is to actually get the required data. The following code snippet shows how to build a query string and how to get the data.

```
StringBuffer query = new StringBuffer();
query.append("select tcr from KlarosTestCaseResult tcr");
List<?> tcr = context.executeQuery(query.toString());
```

The data is returned in a `List` object that must be stored in the context so that it can later be accessed from the report template. The code snippet below shows how to store the list in the context. For more information on building queries please consult the HQL documentation.

```
context.add("results", tcr);
```

The `List` object is stored in the context with the name `results` and can be accessed from the report template by this name. If more data is required, execute more queries to retrieve the data or process the already retrieved data and store the processed data in the context with a different name.

Note

It is possible to store more than one object in the context. Just use a different name for each object.

11.3.2. Creating a SeamPDF Report Template

The code snippets presented in this section show how to build a report template for Klaros-Testmanagement. For detailed information please refer to SeamPDF manual. More information on the Klaros-Testmanagement object model can be found in de.verit.klaros.core.model.

The following snippet shows how to build the frame for the report template. Inside this frame all other Report details can be described and grouped in chapters and sections.

```
<p:document xmlns:ui="http://java.sun.com/jsf/facelets" xmlns:f="http://java.sun.com/jsf/core"
 xmlns:p="http://jboss.org/schema/seam/pdf" title="Klaros-Testmanagement Test Suite Report"
 marginMirroring="true" author="#{user.name}" creator="#{user.name}" pageSize="A4">...
</p:document>
```

Note the usage of the de.verit.klaros.core.model.KlarosUser parameter from the context, `#{user.name}`.

The next code snippet shows how to define headers and footers for all pages in the report. This snippet makes use of the date and the de.verit.klaros.core.model.KlarosUser object in the context.

```
<f:facet name="header">
  <p:font size="8">
    <p:header borderWidthBottom="0.1" borderColorBottom="black" borderWidthTop="0" alignment="center">
      <p:text value="Example report - generated #{date} by #{user.name}"/>
    </p:header>
    <p:footer borderWidthTop="0.1" borderColorTop="black" borderWidthBottom="0" alignment="center">
      <p:text value="Page " />
      <p:pageNumber />
    </p:footer>
  </p:font>
</f:facet>
```

Next the front page for the report should be defined to provide a short summary of the report. To keep this example short only a fragment is presented. For the complete script please see Section E.3, "Example report template".

```
<p:font style="bold" size="16">
  <p:paragraph alignment="center" spacingAfter="5">
    <p:text value="#{user.name} (#{user.email})"/>
  </p:paragraph>
</p:font>
```

The snippet shows how to insert the user's email address which opens the email client when clicked.

11.3.3. Creating a Chart

To provide a graphical overview it is sometimes necessary to add a chart to the report. The required data can be prepared by the Groovy script and stored in the context. Then the

report template can pass the data to the charting component of SeamPDF. This section explains how to create a chart as shown in Figure 11.7, “A Pie Chart Example”.

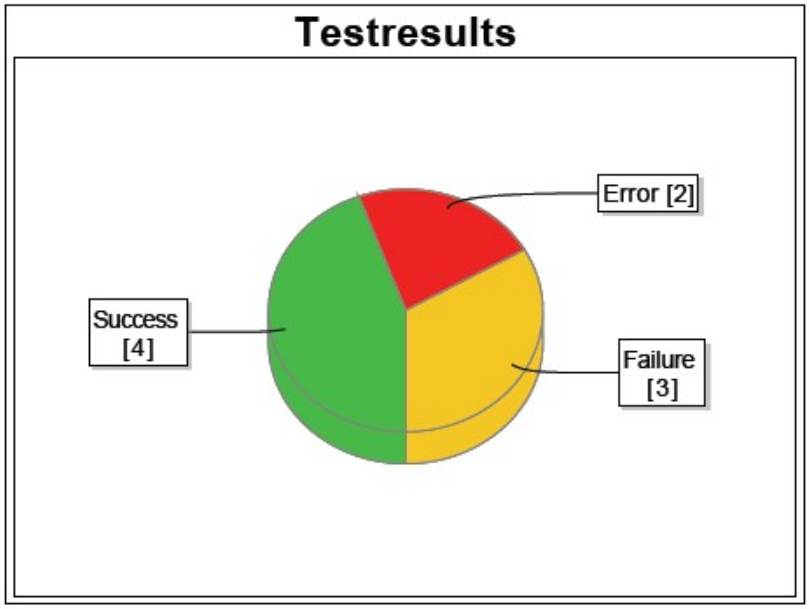


Figure 11.7. A Pie Chart Example

11.3.3.1. Pie Chart Groovy Script

As mentioned before, the Groovy script is not only used to retrieve the data, but it can also be used to prepare the data for the report. The following listing shows a possible way to prepare the data for a pie chart. For every possible result a `List` object is created, then the de.verit.klaros.core.model.KlarosTestCaseResult retrieved before are stored in one of the lists depending on the test result. Next the lists are added to the context.

```
List<KlarosTestCaseResult> error = new ArrayList<KlarosTestCaseResult>();
List<KlarosTestCaseResult> failure = new ArrayList<KlarosTestCaseResult>();
List<KlarosTestCaseResult> success = new ArrayList<KlarosTestCaseResult>();
// Iterate over the results and retrieve the status
Iterator<KlarosTestCaseResult> iter = (Iterator<KlarosTestCaseResult>) tcr.iterator();
while (iter.hasNext()) {
  KlarosTestCaseResult result = iter.next();
    if (result.isError()) error.add(result);
    else if (result.isFailure()) failure.add(result);
    else if (result.isPassed()) success.add(result);
}
context.add("error", error);
context.add("failure", failure);
context.add("success", success);
```

The de.verit.klaros.core.model.KlarosTestCaseResult are split into three lists depending on the test case result. These lists can then be accessed from the context by their corresponding key.

11.3.3.2. Pie Chart Template Script

The following snippet shows how to display the data that was prepared by the Groovy script before.

```
<p:paragraph horizontalAlignment="center">
  <p:piechart title="Testresults" direction="anticlockwise" circular="true"
    startAngle="30" labelGap="0.1" labelLinkPaint="black" plotBackgroundPaint="white"
    labelBackgroundPaint="white" is3D="true">
    <p:series key="results">
      <p:data key="Error [#{error.size}]" value="#{error.size}" sectionPaint="#FF0A0A" />
      <p:data key="Success [#{success.size}]" value="#{success.size}" sectionPaint="#33CC00"/>
      <p:data key="Failure [#{failure.size}]" value="#{failure.size}" sectionPaint="#FFCC00"/>
    </p:series>
  </p:piechart>
</p:paragraph>
```

The `piechart` element builds the frame for the chart by defining the main outlook of the chart. For detailed information on different charts please check the SeamPDF documentation. For a pie chart a series of data is required. The data is retrieved from the lists stored in the context by the Groovy script.

```
<p:data key="Error [#{error.size}]" value="#{error.size}" sectionPaint="#FF0A0A" />
```

This code retrieves the list containing the error results from the context and calls its `size()` method to determine the amount of erroneous test cases. The pie chart is then rendered from the three datasections, as seen in Figure 11.7, "A Pie Chart Example".

11.3.4. Including Images

This section shows how to include an image into a report. Since the report is rendered by Seam, the image to be included must be accessible from Seam. This can either be achieved by storing the image in a .jar file and storing this file in the `.klaros/resources` folder of the user running Klaros-Testmanagement, or by providing the image on a web server, from where it can be included by using HTML code in the report template.

When storing the image in a .jar file it can be easily accessed by the following code snippet.

```
<p:image value="images/image.png"/>
```

The `value` attribute defines the image filename and the folder of the image location.

When providing an image via a web server it can be accessed by the following code snippet.

```
<p:html>
  <img src="http://www.verit.de/images/logo-klaros-160.png" />
</p:html>
```

In this case the `src` attribute defines the URL of the image location.

11.3.5. Creating a SeamExcel Report Template

The code snippets presented in this section show how to build a report template for Klaros-Testmanagement. For detailed information please refer to SeamExcel manual. More information on the Klaros-Testmanagement object model can be found in de.verit.klaros.core.model .

The following snippet shows how to build the frame for the report template. Inside this frame all other report details can be described and grouped in chapters and sections.

```
<e:workbook xmlns:e="http://jboss.org/schema/seam/excel"
  xmlns:f="http://java.sun.com/jsf/core" xmlns:ui="http://java.sun.com/jsf/facelets">
```

```
      <e:worksheet name="#{user.name}" value="#{testCases}" var="testCase">...
      </e:worksheet>
</e:workbook>
```

Note the usage of the de.verit.klaros.core.model.KlarosUser and the de.verit.klaros.core.model.KlarosTestCase parameter from the context, `#{user.name}` and `#{testCases}`.

A worksheet uses a list of objects to generate the rows and columns of the excel sheet. This list must be passed to the `value` attribute of the `worksheet` tag. In the `var` attribute the name of the current object can be defined. For example: To use the test case name of the current object inside the worksheet block the statement `#{testCase.name}` must be used. It is also possible to use multiple `worksheet` tags in one template. In this example we have only one worksheet with the user name as worksheet name.

Chapter 12. Import/Export

Klaros-Testmanagement provides several interfaces to import data from other tools and export its data into several formats.

12.1. Importing Test Cases from Excel

Klaros-Testmanagement allows importing of test cases from Excel sheets structured in a predefined format which will be further detailed in Section 12.1.1, “Import Format”.

In addition to the standard fields provided for test cases users of the Klaros-Testmanagement Enterprise Edition are also able to import data for user-defined fields. For this it is necessary to prepare the project according to what is listed in Section 12.1.2, “Prerequisites”.

The import interface consists of a simple REST interface which can be accessed by command line tools as well as custom applications. This is further detailed in Section 12.1.3, “Execution”.

12.1.1. Import Format

The Excel sheet used to import test cases has to follow this strict format:

1. Each test case is located on a separate sheet.

2. Each sheet is divided into three sections: general properties, test case steps and custom properties.

	A	B	C	D	E	F	G	H
1	ID	TC00051					Cycle	Major
2	Name	Test Login					Category	Regression Tests
3	Revision	1.0						
4								
5	Description	Check the basic login functionality						
6	Precondition	Server installed						
7	Postcondition	User is logged in						
8	Expected Result							
9								
10	Note	Do not skip this!						
11	Area	REGRESSION						
12	Design Method	BLACK_BOX						
13	Variety	NEGATIVE						
14	Execution	MANUAL						
15	Priority	MEDIUM						
16	State							
17	Team	QA						
18	Level	COMPONENT						
19	Document Base	Requirements Document 1.5 (12.7.2011)						
20	Dependency	-						
21	Evaluation	MANUAL						
22	Traceability	UC-112						
23								
24								
25	Step	Action	Precondition	Postcondition	Expected Result			
26	1	Open login page			Login page displayed			
27	2	Enter username						
28	3	Enter password			Password should not be visible			
29	4	Press ok			User is logged in			
30								

Figure 12.1. Test Case Excel Sheet Sample

12.1.1.1. General Properties

The table below lists the cell coordinates for the general test case properties. None of these values are mandatory. Some fields can take only predefined values which are listed there.

Coordinate	Value
B2	Name
B3	Revision (only used in export)
B5	Description
B6	Precondition
B7	Postcondition
B8	Expected Result
B10	Note
B11	Area (`FUNCTIONAL`, `NON_FUNCTIONAL`, `STRUCTURAL`, `REGRESSION`, `RE_TEST`)
B12	Design Method (`BLACK_BOX`, `WHITE_BOX`)
B13	Variety (`POSITIVE`, `NEGATIVE`)
B14	Execution (`MANUAL`, `AUTO`)
B15	Priority (`LOW`, `MEDIUM`, `HIGH`)
B16	State (`NEW`, `APPROVED`, `LOCKED`, `INVISIBLE`)
B17	Team
B18	Level (`COMPONENT`, `INTEGRATION`, `SYSTEM`, `ACCEPTANCE`)
B19	Document Base
B20	Dependency
B21	Evaluation
B22	Traceability

Table 12.1. General Property Coordinates

12.1.1.2. Test Case Steps

Test cases may contain an variable number of test steps. These are listed one per row starting from cell A26 downwards. The import parser will stop reading steps once it encounters an empty step number.

Coordinate	Value
A27 (ff)	Step Number
B27 (ff)	Action
C27 (ff)	Precondition
D27 (ff)	Postcondition
E27 (ff)	Expected Result

Table 12.2. Test Case Step Coordinates

12.1.1.3. Custom Properties

Test cases may contain custom attributes. These are listed as name/value pairs starting from cell G1 downwards.

The name in the name column must exactly match the name of general property for a value to be parsed correctly. The parsing stops once an empty name cell has been found. The value field for enumeration properties must exactly match the name of the enumeration property value definition to be parsed.

Coordinate	Value
G1 (ff)	Property name
H1 (ff)	Property value

Table 12.3. User Defined Property Coordinates

12.1.2. Prerequisites

To import Excel based test cases the following prerequisites have to be met:

1. The file that is going to be imported has to be in *.xls format.
2. The project that should contain the imported test cases has to be already created.
3. If the "Authentication required for import?" flag is set (see Section 10.5.1, “Miscellaneous Settings”), username and password of a system user have to be provided (see Section 10.3.3, “User Details” for more information on system users).
4. If custom properties are to be used they have to be defined for the project prior to starting the import process.

   Please note that custom properties are only supported in Klaros-Testmanagement Enterprise Edition.

12.1.3. Execution

To import Excel-based test cases a REST interface is available which allows you to upload your data from the command-line or other applications.

The following example shows how to import an Excel sheet containing test cases into the project named P00001 using the curl command line application. Curl should be available in every Linux distribution and as part of the Cygwin http://www.cygwin.com/ distribution or as a command line tool from http://curl.haxx.se/download.html for the Microsoft Windows operating system family.

```
curl -v -T TestCases.xls "<klaros-app-url>/seam/resource/rest/import/testcase/xls?config=P00001"
```

Example 12.1. Excel Test Case Import via Command Line

If the Klaros-Testmanagement installation requires authentication for import, username and password arguments have to be appended to the curl call.

```
curl -v -T TestCases.xls "<klaros-app-url>/seam/resource/rest/import/testcase/xls?config=P00001\
&username=user&password=secret"
```

Example 12.2. Excel Test Case Import via Command Line with authentication

Note

All artifacts which are referenced during import (like projects or test cases) contain five digits in their id. For example, *P00001* is a valid project ID, while *P0001* and *P000001* are not.

12.2. Importing Test Cases from XML

Klaros-Testmanagement allows importing of test cases from XML Files. The import process will always create new instances of the supplied test cases for each invocation.

The format of the XML file is described in Appendix C, *Test Case Import File Specification*. For validation purposes an XML schema is also available from the following URL: http://www.klaros-testmanagement.com/files/schema/klaros-testcases-1.0.xsd.

12.2.1. Prerequisites

To import XML based test cases the following prerequisites have to be met:

1. The file that is going to be imported has to be in *.xml format.
2. The project that should contain the imported test cases has to be already created.
3. If the "Authentication required for import?" flag is set (see Section 10.5.1, “Miscellaneous Settings”), username and password of a system user have to be provided (see Section 10.3.3, “User Details” for more information on system users).
4. If custom properties are to be used they have to be defined for the project prior to starting the import process.

   Please note that custom properties are Only available in the Klaros-Testmanagement Enterprise Edition

12.2.2. Execution

To import XML-based test cases a REST interface is available which allows you to upload your data from the command-line or other applications.

The following example shows how to import an XML file containing test cases into the project named P00001 using the curl command line application. Curl should be available in every decent Linux distribution and as part of the Cygwin http://www.cygwin.com/ distribution or as a command line tool from http://curl.haxx.se/download.html for the Microsoft Windows operating system family.

```
curl -v -T TestCases.xml "<klaros-app-url>/seam/resource/rest/import/testcase/xml?config=P00001"
```

Example 12.3. XML Test Case Import via Command Line

If the Klaros-Testmanagement installation requires authentication for importing test cases, username and password arguments have to be appended to the curl call.

```
curl -v -T TestCases.xml "<klaros-app-url>/seam/resource/rest/import/testcase/xml?config=P00001\
&username=user&password=secret"
```

Example 12.4. XML Test Case Import via Command Line with authentication

Note

All artifacts which are referenced during import (like projects or test cases) contain five digits in their id. For example, *P00001* is a valid Project id, while *P01* and *P001* are not.

12.3. Importing Requirements from Excel

Klaros-Testmanagement allows importing of requirements from Excel sheets structured in a predefined format which will be further detailed in Section 12.3.1, “Import Format”.

In addition to the standard fields provided for requirements users of the Klaros-Testmanagement Enterprise Edition are also able to import data for user-defined fields. For this it is necessary to prepare the project according to what is listed in Section 12.3.2, “Prerequisites”.

The import interface consists of a simple REST interface which can be accessed by command line tools as well as custom applications. This is further detailed in Section 12.3.3, “Execution”.

12.3.1. Import Format

The Excel sheet used to import requirements has to follow this strict format:

1. Each requirement is located on a separate sheet.

2. Each sheet is divided into two sections: general properties and custom properties.

	A	B	C	D	E	F	G	H
1	**Shortname**	Test Requirement					**Department**	Internal
2	**Revision**	1.0					**Category**	Security
3	**Description**	Unauthorized users cannot access the server					**Dev.**	BETA
4	**Priority**	HIGH						
5	**Summary**	During alpha development phase, it is possible to access the server without providing a user/password combination. The Server should be secured against unauthorized access during beta phase.					User Defined Properties	
6								
7		General Properties						
8								
9								

Figure 12.2. Requirement Excel Sheet Sample

12.3.1.1. General Properties

The table below lists the cell coordinates for the general requirement properties. None of these values are mandatory. Some fields can take only predefined values which are listed there.

Coordinate	Value
B1	Name
B2	Revision (only used in export)
B3	Description
B4	Priority (`LOW`, `MEDIUM`, `HIGH`)
B5	Summary

Table 12.4. General Property Coordinates

12.3.1.2. Custom Properties

Requirements may contain an arbitrary number of custom attributes. These are listed as name/value pairs starting from cell G1 downwards.

The name in the name column must exactly match the name of general property for a value to be parsed correctly. The parsing stops once an empty name cell has been found. The value field for enumeration properties must exactly match the name of the enumeration property value definition to be parsed.

Coordinate	Value
G1 (ff)	Property name
H1 (ff)	Property value

Table 12.5. User Defined Property Coordinates

12.3.2. Prerequisites

To import Excel based requirements the following prerequisites have to be met:

1. The file that is going to be imported has to be in *.xls format.

2. The project that should contain the imported requirements has to be already created.

3. If the *Authentication required for import?* flag is set (see Section 10.5.1, “Miscellaneous Settings”), username and password of a system user have to be provided (see Section 10.3.3, “User Details” for more information on system users).

4. If custom properties are to be used they have to be defined for the project prior to starting the import process.

   Please note that custom properties are only supported in the Klaros-Testmanagement Enterprise Edition

12.3.3. Execution

To import Excel-based requirements a REST interface is available which allows you to upload your data from the command-line or other applications.

The following example shows how to import an Excel sheet containing requirements into the project named P00001 using the curl command line application. curl is available in almost any Linux distribution and for the Microsoft Windows operating system family as part of the Cygwin http://www.cygwin.com/ distribution or as a command line tool from http://curl.haxx.se/download.html.

```
curl -v -T Requirements.xls "<klaros-app-url>/seam/resource/rest/import/requirement/xls?config=P00001"
```

Example 12.5. Excel Requirement Import via Command Line

If the Klaros-Testmanagement installation requires authentication for import, username and password arguments have to be appended to the curl call.

```
curl -v -T Requirements.xls "<klaros-app-url>/seam/resource/rest/import/requirement/xls?config=P00001\
&username=user&password=secret"
```

Example 12.6. Excel Requirement Import via Command Line with authentication

Note

All artifacts which are referenced during import (like projects or test cases) contain five digits in their id. For example, *P00001* is a valid project ID, while *P01* and *P001* are not.

12.4. Importing and Synchronizing Requirements from XML

Klaros-Testmanagement allows to both import and synchronize requirements from XML Files. An import is limited to a one time action which will always create new instances of the supplied requirements. On the other hand synchronisation allows to update existing requirements with the content in the XML file and create new revisions of them when needed.

The format of the XML file is described in Appendix D, *Requirement Import File Specification*. For validation purposes an XML schema is also available from the following URL: http://www.klaros-testmanagement.com/files/schema/klaros-requirements-1.0.xsd.

12.4.1. Prerequisites

To import XML based requirements the following prerequisites have to be met:

1. The file that is going to be imported has to be in XML format matching the given XML schema.

2. The project that should contain the imported requirements has to be already created.

3. If the *Authentication required for import?* flag is set (see Section 10.5.1, "Miscellaneous Settings"), username and password of a system user have to be provided (see Section 10.3.3, "User Details" for more information on system users).

4. If custom properties are to be used they have to be defined for the project prior to starting the import process.

   Please note that custom properties are only supported in the Klaros-Testmanagement Enterprise Edition.

12.4.2. Importing

To import XML-based requirements a REST interface is available which allows you to upload your data from the command-line or other applications.

The following example shows how to import an XML file containing requirements into the project named P00001 using the curl command line application. curl is available in almost any Linux distribution and for the Microsoft Windows operating system family as part of the Cygwin http://www.cygwin.com/ distribution or as a command line tool from http://curl.haxx.se/download.html.

```
curl -v -T Requirements.xml "<klaros-app-url>/seam/resource/rest/import/requirement/xml?config=P00001"
```

Example 12.7. XML Requirement Import via Command Line

If the Klaros-Testmanagement installation requires authentication for import, username and password arguments have to be appended to the curl call.

```
curl -v -T Requirements.xml "<klaros-app-url>/seam/resource/rest/import/requirement/xml?config=P00001\
&username=user&password=secret"
```

Example 12.8. XML Requirement Import via Command Line with authentication

Note

All artifacts which are referenced during import (like projects or test cases) contain five digits in their ID. For example, *P00001* is a valid project ID, while *P01* and *P001* are not.

12.4.3. Synchronizing

To synchronize XML-based requirements a different REST interface is available (ending with `/sync/requirement/xml`). It understands the same import format as defined for importing requirements but requires additional elements to be present for correct operation.

In contrast to an import a synchronization action is intended to update the set of requirements stored in Klaros-Testmanagement from arbitrary external applications on a regular basis. The synchronization supports creating and updating as well as revisioning of require-

ments. Changes to the requirements will be reflected in the Klaros-Testmanagement database on a per field basis for each synchronization action.

Synchronization overrides local changes!

Be aware that all data received via synchronization will overwrite any changes made to the requirements stored in Klaros-Testmanagement.

Required Synchronization Element `externalId`

Supplying the `externalId` element (See Section D.7, “<externalId>”) will identify a requirement for subsequent synchronisation attempts. It is required field when using this interface. If you change this value, a new requirement object will be created instead of replacing the content of the previously created requirement.

Optional Synchronization Element `externalRevision`

Supplying the `externalRevision` element (See Section D.8, “<externalRevision>”) will identify a new revision of a requirement for subsequent synchronisation attempts. It is optional when using this interface. If you add this value and it does not match a previous revision of yet existing requirement matching the supplied `externalId`, a new requirement revision will be created instead of replacing the content of the existing requirement.

The following example shows how to synchronize an XML file containing requirements into the project named P00001 using the curl command line application. curl is available in almost any Linux distribution and for the Microsoft Windows operating system family as part of the Cygwin http://www.cygwin.com/ distribution or as a command line tool from http://curl.haxx.se/download.html.

```
curl -v -T Requirements.xml "<klaros-app-url>/seam/resource/rest/sync/requirement/xml?config=P00001"
```

Example `Requirements.xml` input:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<r:container xmlns:r="http://klaros-testmanagement.com/export-requirement-1.0">
 <r:requirements>
  <r:requirement>
   <r:attributes/>
   <r:externalId>RTM-00001</r:externalId>
   <r:externalRevision>1.0</r:externalRevision>
   <r:priority>MEDIUM</r:priority>
   <r:shortname>Remote-controlled door panels / unlocking</r:shortname>
   <r:summary>Doors must me remote controllable.</r:summary>
  </r:requirement>
 </r:requirements>
</r:container>
```

Example 12.9. XML Requirement Synchronization via Command Line

If the Klaros-Testmanagement installation requires authentication for import, username and password arguments have to be appended to the curl call as shown below.

```
curl -v -T Requirements.xml "<klaros-app-url>/seam/resource/rest/import/requirement/xml?config=P00001\
&username=user&password=secret"
```

Example 12.10. XML Requirement Import via Command Line with authentication

Note

All artifacts which are referenced during import (like projects or test cases) contain five digits in their ID. For example, *P00001* is a valid project ID, while *P01* and *P001* are not.

12.5. Importing Test Results

Klaros-Testmanagement is able to import test results via a REST interface into the Klaros-Testmanagement database.

12.5.1. Prerequisites

To import test results the following prerequisites have to be met:

1. The project, iteration, test environment and system under test that should contain the imported test case or test suite results have to be created.

2. If the *Authentication required for import?* flag is set (see Section 10.5.1, “Miscellaneous Settings”), username and password of a system user have to be provided (see Section 10.3.3, “User Details” for more information on system users).

12.5.2. Execution

The URL of the import interface is located at `http://${host}:${port}/klaros-web/importer`. The content will be transferred via a HTTP PUT request using the above URL and various URL Query parameters.

The following parameters are supported:

config	The ID of the project to import the results into (e.g. P0001).
iteration	The ID of the iteration to relate the results to (e.g. I00001). This parameter is optional.
env	The ID of the test environment in which the tests have been run (e.g. ENV00001). Please make sure that this test environment already exists in the project before starting the import.
sut	The ID of the system under test in which the tests have been run (e.g. SUT00001). Please make sure that this sys-

	tem under test already exists in the project before starting the import.
type	The type of the import format. The following types are supported:

`aunit`	AUnit
`boost`	Boost Test
`Check`	Check
`cpptest`	CppTest
`cppunit`	CppUnit
`ctest`	ctest
`cunit`	CUnit
`fpunit`	Free Pascal Unit
`jubula`	Jubula / GUIdancer
`junit`	JUnit Selenium TestNG
`mbunit`	MBUnit
`mstest`	MSTest
`nunit`	NUnit
`phpunit`	PHPUnit
`qftest`	QF-Test
`qtestlib`	QTestLib
`ranorex`	Ranorex
`testcomplete`	TestComplete
`tusar`	TUSAR
`cpptestunit`	UnitTest++
`valgrind`	Valgrind
`xUnit.net`	xUnit.net

time	The time of the import. Please make sure the fixed format for the time is `dd.MM.yyyy_HH:mm`.
createTestSuiteResults	If set to `true` test suite results are automatically created for the corresponding test suite information contained in

	the result file. This information may vary with the import format. Additionally a corresponding test suite is created for the test suite result if it did not yet exist.
username (since plugin version 1.1)	The user name for the import. If Klaros-Testmanagement is configured to use authentication for the import a valid user name must be passed to the importer.
password (since plugin version 1.1)	The password for the import. If Klaros-Testmanagement is configured to use authentication for the import a valid password must be passed to the importer.

A complete example for a QF-Test import URL would look like this:

```
http://localhost:18080/klaros-web/seam/resource/rest/importer?\
config=P00001&env=ENV00001&sut=SUT00001&type=qftest&time=01.03.2011_12:00&username=me&password=secret
```

Example 12.11. QF-Test import URL sample

The <klaros-app-url> Term

The http://localhost:18080/klaros-web URL shown above is the default klaros application url when accessing Klaros-Testmanagement from the host it is installed on and may vary with your setup. Throughout this this chapter the term <klaros-app-url> will be used instead to reflect this.

The result file is contained in the HTTP request body.

The **curl** command line tool can be used on Linux or Windows/Cygwin to trigger an import in a single command line.

```
curl -v -H "Content-Type: text/xml" -T <test result file> \
"<klaros-app-url>/seam/resource/rest/importer?config=P00001&env=ENV00001&sut=SUT00001&type=junit\
&time=23.05.2011_14:55"
```

Example 12.12. curl command line example

Note

All artifacts which are referenced during import (like projects or test cases) contain five digits in their ID. For example, *P00001* is a valid project ID, while *P01* and *P001* are not.

12.5.2.1. Jenkins/Hudson Plugin for Klaros-Testmanagement

This plugin integrates a continuous integration server with Klaros-Testmanagement by publishing the test results of a Jenkins/Hudson build to the Klaros-Testmanagement application. The test results will be stored in the Klaros-Testmanagement database for further evaluation and reporting purposes. You can find the installation and configuration guide for the plugin in the Jenkins Wiki or Hudson Wiki.

12.6. Exporting Table Content to Excel

Feature only available in Klaros-Testmanagement Enterprise Edition

With Klaros-Testmanagement, it is possible to export the content of all tables to an Excel file. The current filter and sort settings will be considered.

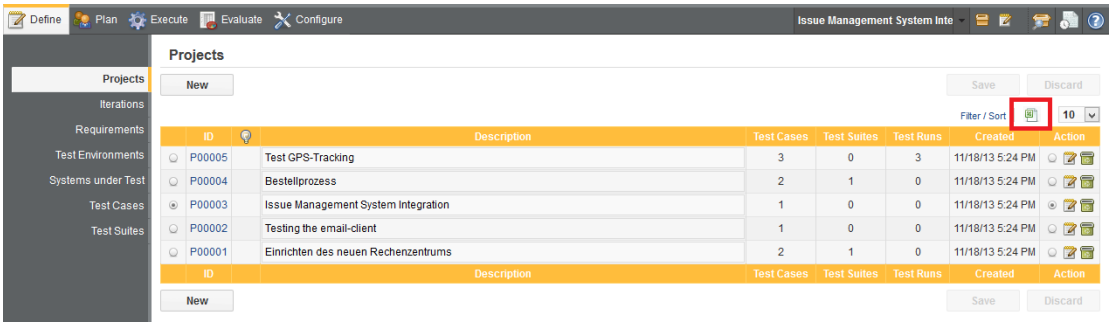


Figure 12.3. Export Table Content to Excel

12.7. Backup/Recovery

To move data between different database installations, or to selectively import data, Klaros-Testmanagement provides the functionality to import and export database content via XML files. Section 10.7, “Backup” explains the import and export functionality in detail.

Glossary

A

Admin	See Administrator.
Administrator	User role that has access to all functionalities in Klaros-Testmanagement.
Artifact	An Artifact is a definable object like a Project, Iteration, Requirement, Test Environment, System under Test, Job, Test Case, Test Suite or Test Case.

B

Bugzilla	Bugzilla is an open source bug tracking and testing tool.

C

Category	Artifacts can be assigned to any number of user-defined categories. Using Categories Users can group related Artifacts together.

C

Coverage	A database is a collection of information organized into interrelated tables of data and specifications of data objects.
Compliance	

D

Database	A database is a collection of information organized into interrelated tables of data and specifications of data objects.
Defect	See Issue.
Defect Management System	See Issue Management System.

E

E-Mail	Electronic mail, often abbreviated as e-mail, is any method of creating, transmitting, or storing primarily text-based communications with digital communications systems.
Error	An error is the inability of the system to perform a test correctly. Not to be confused with Failure.

F

Failure	A failure is a discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. Not to be confused with Error.

G

Guest	User role that can display artifacts and reports but may not change any data.
GUIdancer	GUIdancer is an Eclipse-based tool for automated functional testing through the Graphical User Interface (GUI).

I

Incident	See Issue.
Incident Management System	See Issue Management System.
Issue	The term issue is a unit of work to accomplish an improvement in a system. An Issue could be a bug, a requested feature, task, missing documentation, and likewise.
Issue Management System	An Issue Management System (Issue Tracking System) is a software to manage Issues.
Issue Tracking System	See Issue Management System.
Iteration	An iteration is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, which grows from iteration to iteration to become the final product (ISTQB glossary).

J

Java	Java is a programming language. Most often, Java is used as a abbreviation for Java Runtime Environment, which needs to be installed in order to run Klaros-Testmanagement.
Java Runtime Environment	The Java Runtime environment needs to be installed in order to execute applications programmed in the Java programming language.
JavaScript	JavaScript is a scripting language most often used to add functionality to web pages. Most newer Web browsers can process JavaScript generated code.
Java Runtime Environment	See Java Runtime Environment.

JIRA — JIRA is a bug tracking, Issue tracking, and project management system by Atlassian Software .

Job — Jobs may consist of the execution of Test Cases, Test Suites or any other possible task. Jobs can be nested and assigned to individual users. The executions and results of jobs can be tracked by Klaros-Testmanagement.

Jubula — Jubula provides automated functional GUI testing for various types of applications.

JUnit — JUnit is a unit testing framework for the Java programming language.

M

Manager — User role that has access to create, edit, delete and search for objects, run test cases and test suites, show results and generate reports.

Mantis — Mantis (MantisBT, Mantis Bug Tracker) is an open source bug tracking system.

O

Operating System — An operating system (commonly abbreviated to either OS or O/S) is an interface between hardware and applications. It is responsible for the management and coordination of activities and the sharing of the limited resources of the computer. Common contemporary operating systems include Microsoft Windows, Mac OS, Linux, BSD and Solaris.

OS — See Operating System.

P

Postcondition — Environmental and state conditions that must be fulfilled after the execution of a test or test procedure.

Precondition — Environmental and state conditions that must be fulfilled before the component or system can be executed with a particular test or test procedure.

Project — A project is the main unit that contains all other Artifacts that are needed to execute Test Cases.

Q

QFTest — QF-Test is a professional tool for automated testing of Java and Web applications with a graphical user interface from Quality First Software.

R

Redmine — Redmine is an open source bug tracking, issue tracking, and project management system.

Requirement — A condition or capability needed by a user to solve a problem or achieve an objective that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document (After IEEE 610).

Role — A role defines the rights the rights a user has regarding the application or selected projects.

In Klaros-Testmanagement a user can take the role of an `Administrator`, `Manager`, `Tester` or `Guest`.

S

Selenium — *Selenium* is a web browser automation tool primarily used for automated testing of web apps. Selenium is able to produce *JUnit*-compatible test results, which can be imported into Klaros-Testmanagement.

SUT — See System under Test.

System Account — A System Account is a user that is not able to login at the login page and interactively control the application. System Accounts may be used for automated tasks like importing test results.

System under Test — A System under Test is used to represent a version of a software product that can be tested.

T

Test Case — A Test Case is a set of input values, execution preconditions, expected results and execution post-conditions, developed for a particular objective or test condition, such as determine whether an application or software system meets its specifications.

Test Case Result — The final verdict on the execution of a test and its outcomes, like pass, fail, or error. The result of error is used for situations where it is not clear whether the problem is in the test object.

Test Case Step — A test case step consists of execution preconditions, expected results and execution post-conditions for a single action during test execution.

Test Environment — Test Environments represent the extrinsic settings that may influence the test result. Examples for components of a Test Environments are the Operating System or an application server (e.g. Tomcat 7 on Ubuntu 12.10).

Tester	User role that has access to display artifacts and reports and run jobs, test cases and test suites.
Test Execution	The process of running a test by the component or system under test, producing actual result(s).
Test Run	The result of running a test case or test suite under the same system under test and test environment giving or more test case result(s).
Test Suite	A test suite is a set of test cases and the test cases can be executed in groups. The test suite will be used to verify and ensure that a product or system meets its design specifications and other requirements.
Trac	Trac is an open source bug tracking and issue tracking system.

U

URL	A Uniform Resource Locator (URL) specifies where an identified resource is available and the mechanism for retrieving it. Examples for URLs are http://www.klaros-testmanagement.com/" or file:///C:/klaros.txt

W

Web browser	A Web browser is a software application which enables a user to display and interact with text, images and other information typically located on a Web page at a Web site on the World Wide Web or a local area network.
Windows	Windows is the name of several Operating Systems by Microsoft.

Appendix A. Role Permission Overview

Actions/Roles	Administrator	Manager	Tester	Guest
Execute test cases	✓	✓	✓	✗
Generate reports	✓	✓	✓	✓
View Test Results	✓	✓	✓	✓
Backup and Restore [a]	✓	✓	✗	✗
Create, edit and delete test artifacts	✓	✓	✗	✗
Create, edit and delete users with the role *Tester* [b]	✓	✓	✗	✗
Create, edit and delete users with the role *Administrator* or *Manager*	✓	✗	✗	✗
Edit system preferences [b]	✓	✗	✗	✗

[a] Global managers are able to restore any project. However, they may only export projects which have no project-specific roles defined or in which they have the role *Manager*.

[b] Not changed by project role

Table A.1. Role Permission Overview Table - Klaros-Testmanagement Community Edition

Actions/Roles	Administrator	Manager	Tester	Guest
Configure dashboard	✓	✓	✓	✓
Execute jobs	✓	✓	✓	✗
Export Excel tables	✓	✓	✓	✓
Generate custom reports	✓	✓	✓	✓
Generic full text search	✓	✓	✓	✓
Track test run execution	✓	✓	✓	✗
Create and assign jobs	✓	✓	✗	✗
Create Custom Reports [a]	✓	✓	✗	✗
Define custom fields	✓	✓	✗	✗
Configure LDAP authentication [a]	✓	✗	✗	✗

[a] Not changed by project role

Table A.2. Role Permission Overview Table - Klaros-Testmanagement Enterprise Edition

Actions/Roles	Administrator	Manager	Tester	Guest
Execute jobs in the App	✓	✓	✓	✗
View jobs in the App	✓	✓	✓	✗

Actions/Roles	Administrator	Manager	Tester	Guest
Enable/Disable mobile devices for the - Klaros-Testmanagement instance [a]				
View mobile devices for the Klaros-Testmanagement instance [a]				

[a]Not changed by project role

Table A.3. Role Permission Overview Table - Klaros-Testmanagement Mobile Edition

Appendix B. Model API Reference

B.1. API-Reference

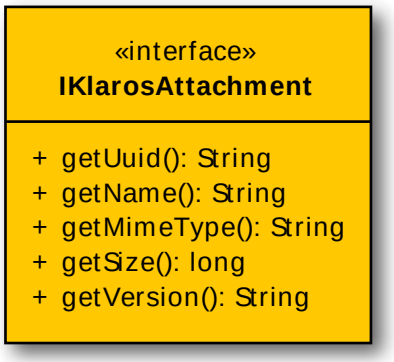
B.1.1. de.verit.klaros.core.model

B.1.1.1. IKlarosAttachment

This interface provides access to a binary attachment.

B.1.1.1.1. Synopsis

```
public interface IKlarosAttachment {

  // Public Methods
  public abstract String getMimeType();
  public abstract String getName();
  public abstract long getSize();
  public abstract String getUuid();
  public abstract String getVersion();
}
```



B.1.1.1.2. getMimeType()

```
public abstract String getMimeType();
```

Gets the mime type of the attachment.

Parameters

`return`	the mime type

B.1.1.1.3. getName()

```
public abstract String getName();
```

Gets the name of the attachment.

Parameters

`return`	the name

B.1.1.1.4. getSize()

```
public abstract long getSize();
```

Gets the size in bytes of this attachment.

Parameters

return	the size

B.1.1.1.5. getUuid()

```
public abstract String getUuid();
```

Get the unique id of this attachment. This may be resolved by calling KlarosContext.getAttachmentURL() to an URL pointing to the attachment data.

Parameters

return	The uuid of this attachment.

B.1.1.1.6. getVersion()

```
public abstract String getVersion();
```

Gets the version of this attachment.

Parameters

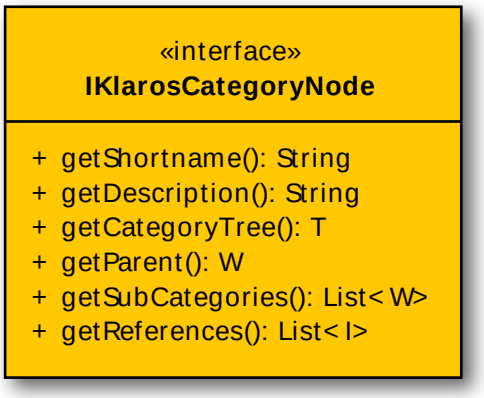
return	the version

B.1.1.2. IKlarosCategoryNode

A Klaros Category Node.

B.1.1.2.1. Synopsis

```
public interface IKlarosCategoryNode<I,T,W,P> {

  // Public Methods
  public abstract T getCategoryTree();
  public abstract String getDescription();
  public abstract W getParent();
  public abstract List<I> getReferences();
  public abstract String getShortname();
  public abstract List<W> getSubCategories();
}
```



B.1.1.2.2. getCategoryTree()

```
public abstract T getCategoryTree();
```

Gets the category tree.

Parameters

`return`	the category tree

B.1.1.2.3. getDescription()

```
public abstract String getDescription();
```

Gets the description.

Parameters

`return`	the description

B.1.1.2.4. getParent()

```
public abstract W getParent();
```

Gets the parent category node.

Parameters

`return`	the parent

B.1.1.2.5. getReferences()

```
public abstract List<I> getReferences();
```

Gets the references of this category.

Parameters

`return`	the references

B.1.1.2.6. getShortname()

```
public abstract String getShortname();
```

Gets the shortname.

Parameters

`return`	the shortname

B.1.1.2.7. getSubCategories()

```
public abstract List<W> getSubCategories();
```

Gets the sub categories.

Parameters

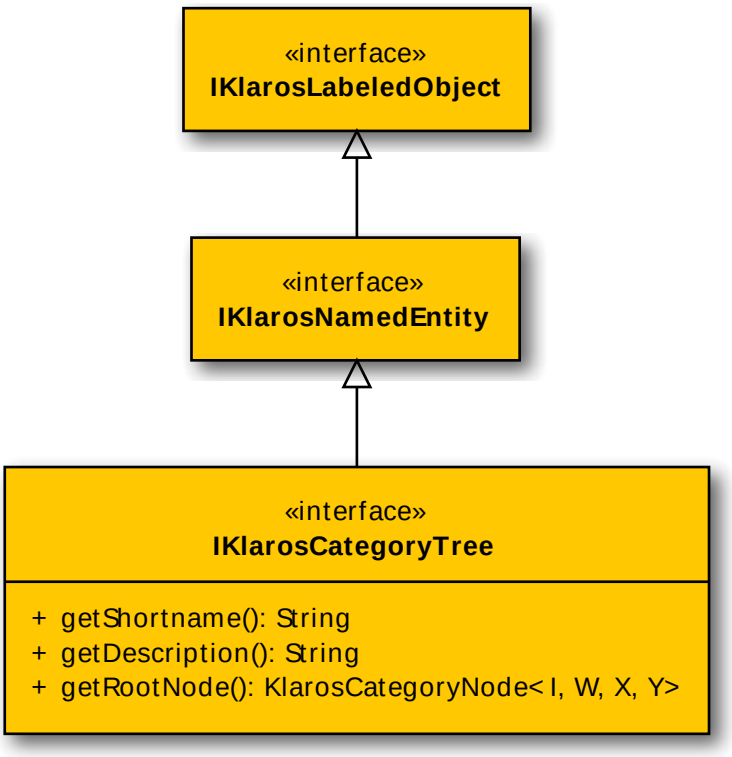
`return`	the sub categories

B.1.1.3. IKlarosCategoryTree

A generic category tree.

B.1.1.3.1. Synopsis

```
public interface IKlarosCategoryTree<I,W extends KlarosCategoryTree<I, W, P, X,
 Y>,P extends CategoryTree,X extends KlarosCategoryNode<I, W, X, Y>,Y extends
 CategoryNode> extends IKlarosNamedEntity {

  // Public Methods
  public abstract String getDescription();
  public abstract KlarosCategoryNode<I, W, X, Y> getRootNode();
  public abstract String getShortname();
}
```



B.1.1.3.2. getDescription()

```
public abstract String getDescription();
```

Gets the description.

Parameters

return	the description

B.1.1.3.3. getRootNode()

```
public abstract KlarosCategoryNode<I, W, X, Y> getRootNode();
```

Gets the root category node.

Parameters

return	the root

B.1.1.3.4. getShortname()

```
public abstract String getShortname();
```

Gets the shortname.

Parameters

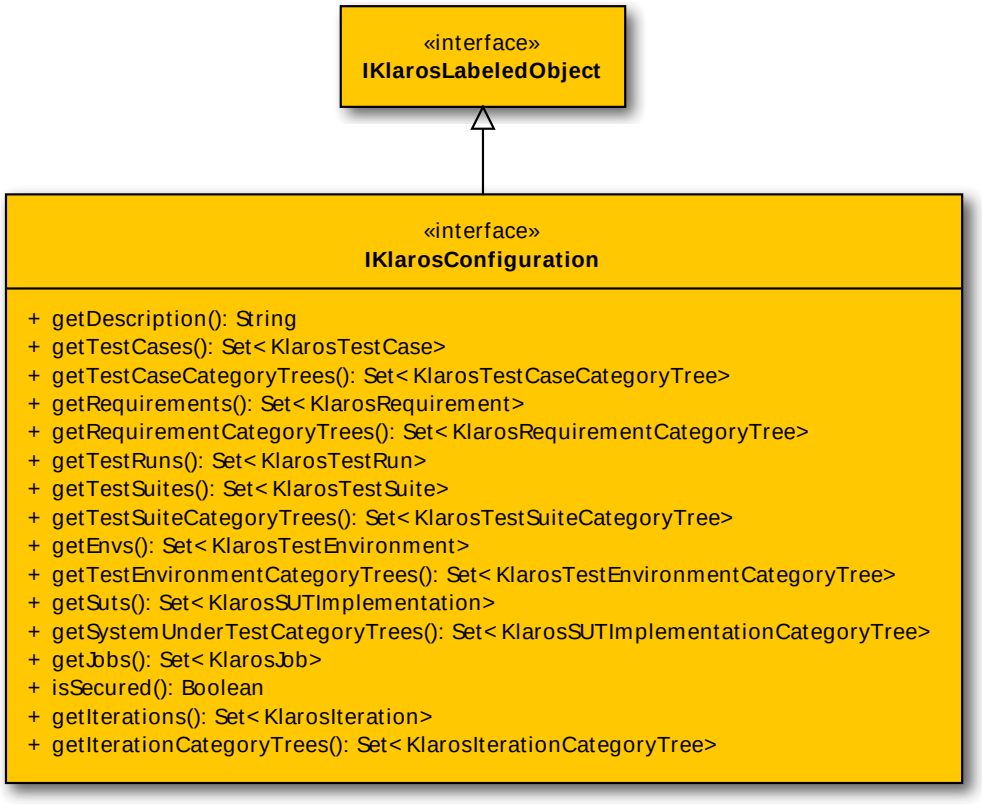
return	the shortname

B.1.1.4. IKlarosConfiguration

This interface provides access to data of a test project.

B.1.1.4.1. Synopsis

```
public interface IKlarosConfiguration extends IKlarosLabeledObject {

  // Public Methods
  public abstract String getDescription();
  public abstract Set<KlarosTestEnvironment> getEnvs();
  public abstract Set<KlarosIterationCategoryTree> getIterationCategoryTrees();
  public abstract Set<KlarosIteration> getIterations();
  public abstract Set<KlarosJob> getJobs();
  public abstract Set<KlarosRequirementCategoryTree> getRequirementCategoryTrees();
  public abstract Set<KlarosRequirement> getRequirements();
  public abstract Set<KlarosSUTImplementation> getSuts();
  public abstract Set<KlarosSUTImplementationCategoryTree> getSystemUnderTestCategoryTrees();
  public abstract Set<KlarosTestCaseCategoryTree> getTestCaseCategoryTrees();
  public abstract Set<KlarosTestCase> getTestCases();
  public abstract Set<KlarosTestEnvironmentCategoryTree> getTestEnvironmentCategoryTrees();
  public abstract Set<KlarosTestRun> getTestRuns();
  public abstract Set<KlarosTestSuiteCategoryTree> getTestSuiteCategoryTrees();
  public abstract Set<KlarosTestSuite> getTestSuites();
  public abstract Boolean isSecured();
}
```



B.1.1.4.2. getDescription()

```
public abstract String getDescription();
```

Returns the project description.

Parameters

`return`	The description of the project.

B.1.1.4.3. getEnvs()

```
public abstract Set<KlarosTestEnvironment> getEnvs();
```

Returns the project test environments.

Parameters

`return`	Set containing the test environments of the project.

B.1.1.4.4. getIterationCategoryTrees()

```
public abstract Set<KlarosIterationCategoryTree> getIterationCategoryTrees();
```

Returns the project requirement category trees.

Parameters

`return`	Set containing the iteration category trees of the project.

B.1.1.4.5. getIterations()

```
public abstract Set<KlarosIteration> getIterations();
```

Returns the project iteration.

Parameters

`return`	KlarosSet containing the iteration objects of the project.

B.1.1.4.6. getJobs()

```
public abstract Set<KlarosJob> getJobs();
```

Returns the project jobs.

Parameters

`return`	KlarosSet containing the job objects of the project.

B.1.1.4.7. getRequirementCategoryTrees()

```
public abstract Set<KlarosRequirementCategoryTree> getRequirementCategoryTrees();
```

Returns the project requirement category trees.

Parameters

`return`	Set containing the requirement category trees of the project.

B.1.1.4.8. getRequirements()

```
public abstract Set<KlarosRequirement> getRequirements();
```

Returns the project requirements.

Parameters

`return`	Set containing the requirement objects of the project.

B.1.1.4.9. getSuts()

```
public abstract Set<KlarosSUTImplementation> getSuts();
```

Returns the project SUTs (systems under test).

Parameters

`return`	KlarosSet containing the SUT objects of the project.

B.1.1.4.10. getSystemUnderTestCategoryTrees()

```
public abstract Set<KlarosSUTImplementationCategoryTree> getSystemUnderTestCategoryTrees();
```

Returns the project system under test category trees.

Parameters

`return`	Set containing the system under test category trees of the project.

B.1.1.4.11. getTestCaseCategoryTrees()

```
public abstract Set<KlarosTestCaseCategoryTree> getTestCaseCategoryTrees();
```

Returns the project test case category trees.

Parameters

return	Set containing the test case category trees of the project.

B.1.1.4.12. getTestCases()

```
public abstract Set<KlarosTestCase> getTestCases();
```

Returns the project test cases.

Parameters

return	Set containing the test case objects of the project.

B.1.1.4.13. getTestEnvironmentCategoryTrees()

```
public abstract Set<KlarosTestEnvironmentCategoryTree> getTestEnvironmentCategoryTrees();
```

Returns the project test environment category trees.

Parameters

return	Set containing the test environment category trees of the project.

B.1.1.4.14. getTestRuns()

```
public abstract Set<KlarosTestRun> getTestRuns();
```

Returns the project test runs.

Parameters

return	Set containing the test run objects of the project.

B.1.1.4.15. getTestSuiteCategoryTrees()

```
public abstract Set<KlarosTestSuiteCategoryTree> getTestSuiteCategoryTrees();
```

Returns the project test suite category trees.

Parameters

return	Set containing the test suite category trees of the project.

B.1.1.4.16. getTestSuites()

```
public abstract Set<KlarosTestSuite> getTestSuites();
```

Returns the project test suites.

Parameters

return	Set containing the test suite objects of the project.

B.1.1.4.17. isSecured()

```
public abstract Boolean isSecured();
```

Returns whether this project is secured or not.

Parameters

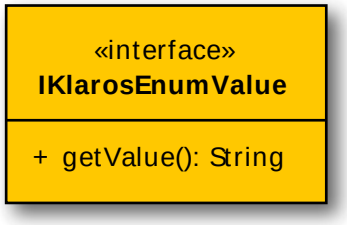
return	true if this project is secured, false if not.

B.1.1.5. IKlarosEnumValue

A Custom property enumeration value.

B.1.1.5.1. Synopsis

```
public interface IKlarosEnumValue {

  // Public Methods
  public abstract String getValue();
}
```



B.1.1.5.2. getValue()

```
public abstract String getValue();
```

Gets the enumeration value.

Parameters

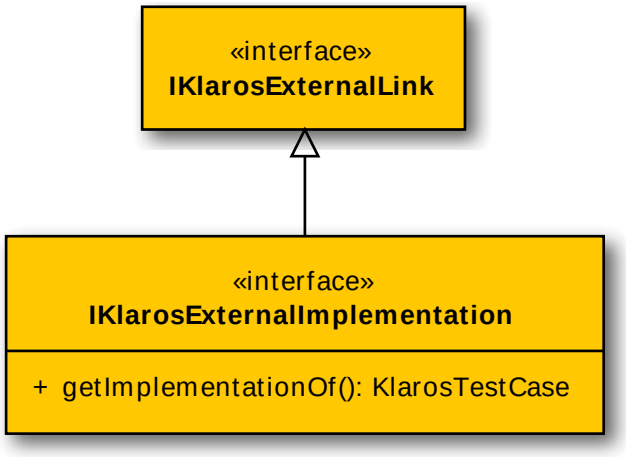
return	the value

B.1.1.6. IKlarosExternalImplementation

This interface provides access to the information about the implementation of a test case.

B.1.1.6.1. Synopsis

```
public interface IKlarosExternalImplementation extends IKlarosExternalLink {

  // Public Methods
  public abstract KlarosTestCase getImplementationOf();
}
```



B.1.1.6.2. getImplementationOf()

```
public abstract KlarosTestCase getImplementationOf();
```

Get related test case.

Parameters

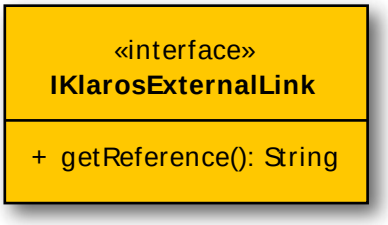
`return`	The test case for which this is an implementation.

B.1.1.7. IKlarosExternalLink

This interface provides access to externally stored information about an object.

B.1.1.7.1. Synopsis

```
public interface IKlarosExternalLink {

  // Public Methods
  public abstract String getReference();
}
```



B.1.1.7.2. getReference()

```
public abstract String getReference();
```

Get the reference to the externally stored information.

Parameters

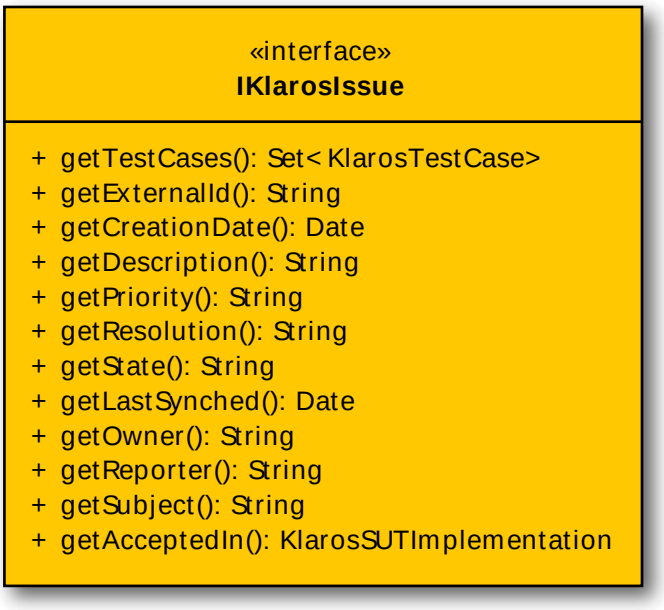
`return`	The reference to the information.

B.1.1.8. IKlarosIssue

This interface provides access to a software issue.

B.1.1.8.1. Synopsis

```
public interface IKlarosIssue {

  // Public Methods
  public abstract KlarosSUTImplementation getAcceptedIn();
  public abstract Date getCreationDate();
  public abstract String getDescription();
  public abstract String getExternalId();
  public abstract Date getLastSynched();
  public abstract String getOwner();
  public abstract String getPriority();
  public abstract String getReporter();
  public abstract String getResolution();
  public abstract String getState();
  public abstract String getSubject();
  public abstract Set<KlarosTestCase> getTestCases();
}
```



B.1.1.8.2. getAcceptedIn()

```
public abstract KlarosSUTImplementation getAcceptedIn();
```

Gets the system under test this issue has been accepted/detected in.

Parameters

`return`	the system under test

B.1.1.8.3. getCreationDate()

```
public abstract Date getCreationDate();
```

Gets the creation date of this issue.-.

Parameters

`return` the creation date of this issue.

B.1.1.8.4. getDescription()

```
public abstract String getDescription();
```

Get the description.

Parameters

`return` The description of this issue.

B.1.1.8.5. getExternalId()

```
public abstract String getExternalId();
```

Gets the external id of this issue, if available. This id is generated by the external issue management system upon creation of the issue.

Parameters

`return` the external id

B.1.1.8.6. getLastSynched()

```
public abstract Date getLastSynched();
```

Gets the last synchronization date of this issue.

Parameters

`return` the last synched date of this issue.

B.1.1.8.7. getOwner()

```
public abstract String getOwner();
```

Gets the name of the owner/assignee of this issue.

Parameters

`return` the owner name of this issue.

B.1.1.8.8. getPriority()

```
public abstract String getPriority();
```

Gets the priority of this issue. The format and content of this value varies with the issue management system and its configuration.

Parameters

`return` the priority of this issue.

B.1.1.8.9. getReporter()

```
public abstract String getReporter();
```

Gets the name of the reporter of this issue.

Parameters

`return` the reporter name of this issue.

B.1.1.8.10. getResolution()

```
public abstract String getResolution();
```

Gets the resolution of this issue. The format and content of this value varies with the issue management system and its configuration.

Parameters

`return` the resolution of this issue.

B.1.1.8.11. getState()

```
public abstract String getState();
```

Gets the state of this issue. The format and content of this value varies with the issue management system and its configuration.

Parameters

`return` the state string of this issue.

B.1.1.8.12. getSubject()

```
public abstract String getSubject();
```

Gets the subject of this issue.

Parameters

`return` the subject string of this issue.

B.1.1.8.13. getTestCases()

```
public abstract Set<KlarosTestCase> getTestCases();
```

Get the related test cases.

Parameters

`return` `Set` of test cases which have detected this issue.

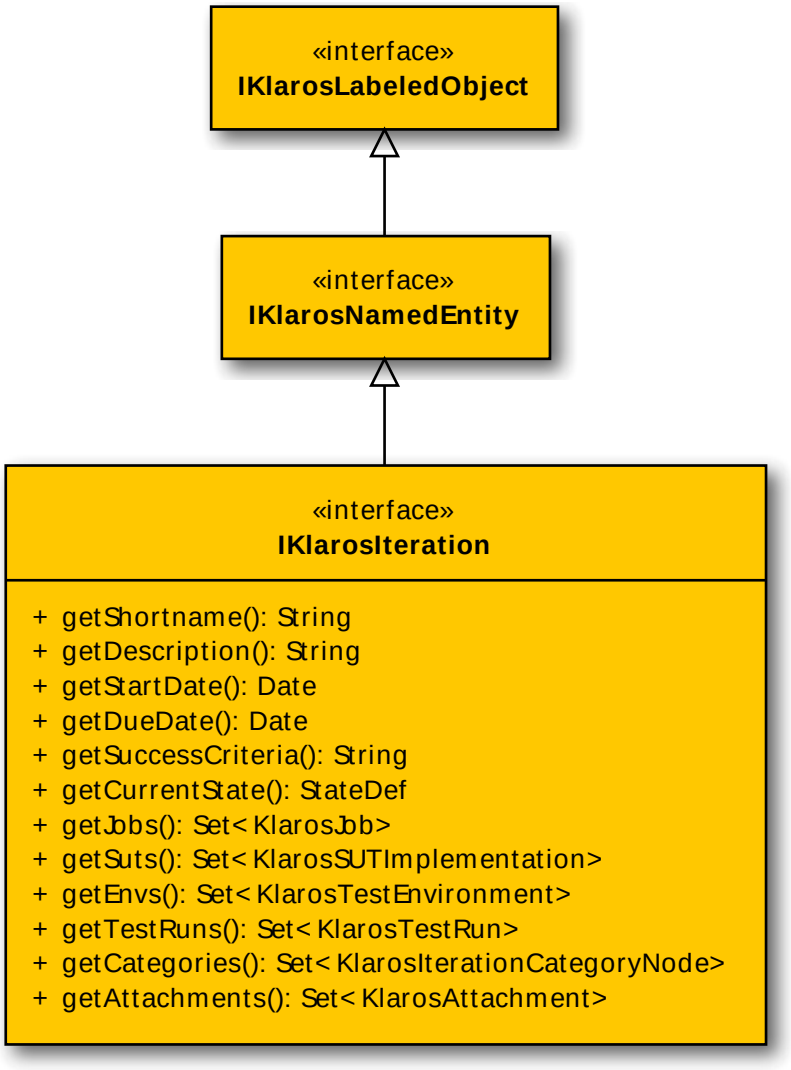
B.1.1.9. IKlarosIteration

An iteration in a project.

B.1.1.9.1. Synopsis

```
public interface IKlarosIteration extends IKlarosNamedEntity {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosIterationCategoryNode> getCategories();
  public abstract StateDef getCurrentState();
  public abstract String getDescription();
  public abstract Date getDueDate();
  public abstract Set<KlarosTestEnvironment> getEnvs();
  public abstract Set<KlarosJob> getJobs();
```

```
  public abstract String getShortname();
  public abstract Date getStartDate();
  public abstract String getSuccessCriteria();
  public abstract Set<KlarosSUTImplementation> getSuts();
  public abstract Set<KlarosTestRun> getTestRuns();
}
```



B.1.1.9.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this iteration.

Parameters

`return`	the attachments

B.1.1.9.3. getCategories()

```
public abstract Set<KlarosIterationCategoryNode> getCategories();
```

Gets the categories this object belongs to. Each category node will belong to a different category tree.

Parameters

`return`	the category nodes

B.1.1.9.4. getCurrentState()

```
public abstract StateDef getCurrentState();
```

Gets the current state.

Parameters

`return`	the current state

B.1.1.9.5. getDescription()

```
public abstract String getDescription();
```

Gets the description.

Parameters

`return`	the description

B.1.1.9.6. getDueDate()

```
public abstract Date getDueDate();
```

Gets the due date.

Parameters

`return`	the due date

B.1.1.9.7. getEnvs()

```
public abstract Set<KlarosTestEnvironment> getEnvs();
```

Gets the test environments associated with this iteration.

Parameters

`return`	the test environments

B.1.1.9.8. getJobs()

```
public abstract Set<KlarosJob> getJobs();
```

Gets the jobs associated with this iteration.

Parameters

`return`	the jobs

B.1.1.9.9. getShortname()

```
public abstract String getShortname();
```

Gets the short name.

Parameters

`return`	the short name

B.1.1.9.10. getStartDate()

```
public abstract Date getStartDate();
```

Gets the start date.

Parameters

return	the start date

B.1.1.9.11. getSuccessCriteria()

```
public abstract String getSuccessCriteria();
```

Gets the success criteria.

Parameters

return	the success criteria

B.1.1.9.12. getSuts()

```
public abstract Set<KlarosSUTImplementation> getSuts();
```

Gets the systems under test associated with this iteration.

Parameters

return	the systems under test

B.1.1.9.13. getTestRuns()

```
public abstract Set<KlarosTestRun> getTestRuns();
```

Gets the test runs associated with this iteration.

Parameters

return	the test runs

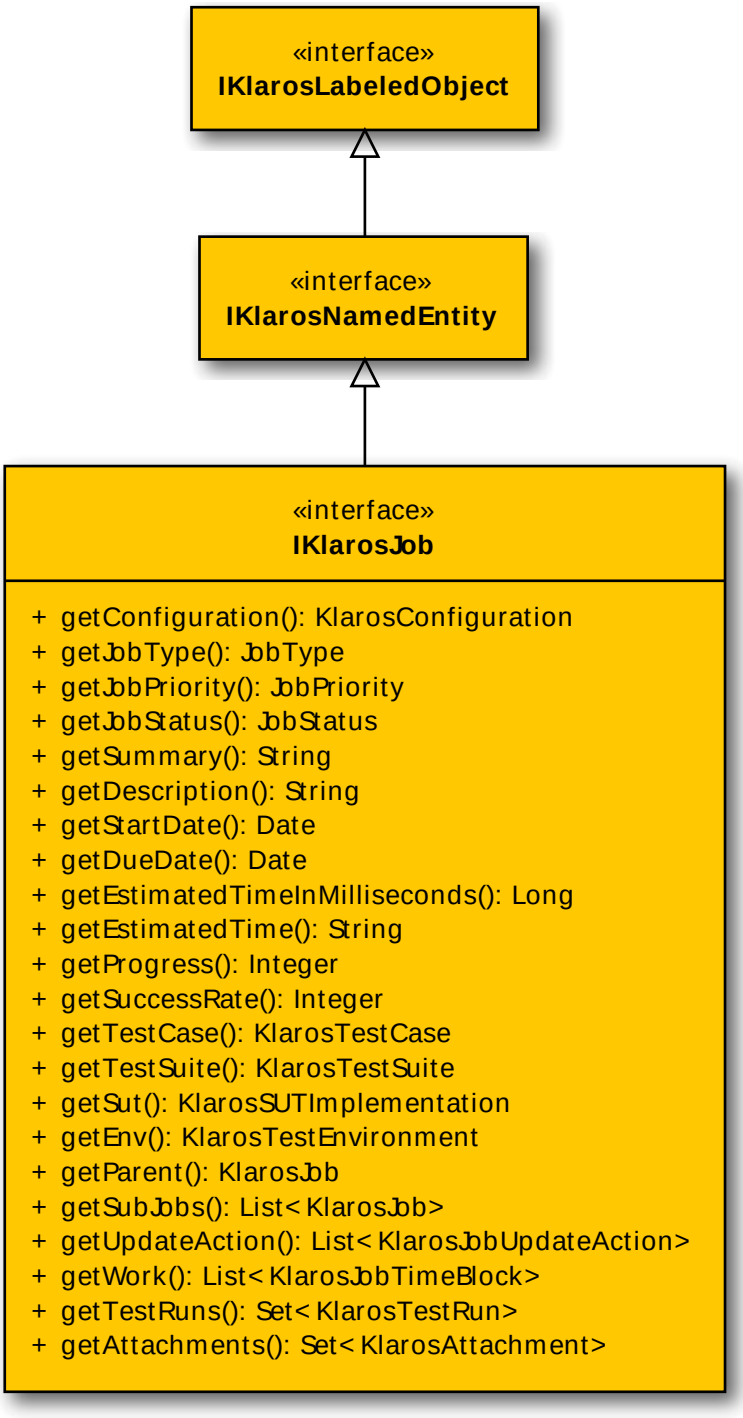
B.1.1.10. IKlarosJob

This interface provides access to the data of a job.

B.1.1.10.1. Synopsis

```
public interface IKlarosJob extends IKlarosNamedEntity {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract KlarosConfiguration getConfiguration();
  public abstract String getDescription();
  public abstract Date getDueDate();
  public abstract KlarosTestEnvironment getEnv();
  public abstract String getEstimatedTime();
  public abstract Long getEstimatedTimeInMilliseconds();
  public abstract JobPriority getJobPriority();
  public abstract JobStatus getJobStatus();
  public abstract JobType getJobType();
  public abstract KlarosJob getParent();
  public abstract Integer getProgress();
  public abstract Date getStartDate();
```

```
  public abstract List<KlarosJob> getSubJobs();
  public abstract Integer getSuccessRate();
  public abstract String getSummary();
  public abstract KlarosSUTImplementation getSut();
  public abstract KlarosTestCase getTestCase();
  public abstract Set<KlarosTestRun> getTestRuns();
  public abstract KlarosTestSuite getTestSuite();
  public abstract List<KlarosJobUpdateAction> getUpdateAction();
  public abstract List<KlarosJobTimeBlock> getWork();
}
```



B.1.1.10.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this job.

Parameters

`return`	the attachments

B.1.1.10.3. getTestRuns()

```
public abstract Set<KlarosTestRun> getTestRuns();
```

Gets the test runs executed with this job.

Parameters

`return`	the test runs

B.1.1.10.4. getUpdateAction()

```
public abstract List<KlarosJobUpdateAction> getUpdateAction();
```

Gets the update action list.

Parameters

`return`	the update action

B.1.1.10.5. getWork()

```
public abstract List<KlarosJobTimeBlock> getWork();
```

Gets the list of work items done on this job.

Parameters

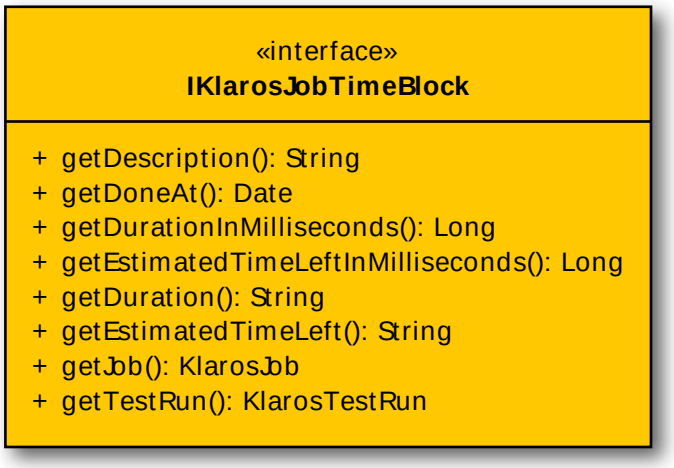
`return`	the work items

B.1.1.11. IKlarosJobTimeBlock

The job time block defines a certain amount of time the job has been worked on.

B.1.1.11.1. Synopsis

```
public interface IKlarosJobTimeBlock {

  // Public Methods
  public abstract String getDescription();
  public abstract Date getDoneAt();
  public abstract String getDuration();
  public abstract Long getDurationInMilliseconds();
  public abstract String getEstimatedTimeLeft();
  public abstract Long getEstimatedTimeLeftInMilliseconds();
  public abstract KlarosJob getJob();
  public abstract KlarosTestRun getTestRun();
}
```



B.1.1.11.2. getDescription()

```
public abstract String getDescription();
```

Gets the work description.

Parameters

`return`	the work description

B.1.1.11.3. getDoneAt()

```
public abstract Date getDoneAt();
```

Gets the date this work was done at.

Parameters

`return`	the start date

B.1.1.11.4. getDuration()

```
public abstract String getDuration();
```

Gets the work duration as a String.

Parameters

`return`	the duration

B.1.1.11.5. getDurationInMilliseconds()

```
public abstract Long getDurationInMilliseconds();
```

Gets the work duration in milliseconds.

Parameters

`return`	the work duration in hours

B.1.1.11.6. getEstimatedTimeLeft()

```
public abstract String getEstimatedTimeLeft();
```

Gets the estimated time left for this job as a String.

Parameters

`return`	the estimated time left

B.1.1.11.7. getEstimatedTimeLeftInMilliseconds()

```
public abstract Long getEstimatedTimeLeftInMilliseconds();
```

Gets the estimated time left for this job in milliseconds.

Parameters

`return`	the estimated time left in hours

B.1.1.11.8. getJob()

```
public abstract KlarosJob getJob();
```

Gets the job this job time block belongs to.

Parameters

`return`	the job

B.1.1.11.9. getTestRun()

```
public abstract KlarosTestRun getTestRun();
```

Gets the test run this job time block belongs to.

Parameters

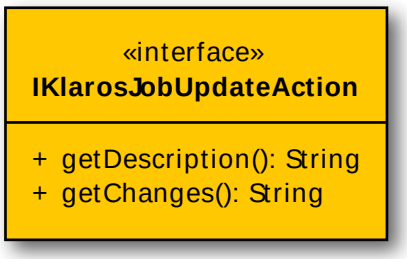
`return`	the test run

B.1.1.12. IKlarosJobUpdateAction

This interface provides access to the job update action.

B.1.1.12.1. Synopsis

```
public interface IKlarosJobUpdateAction {

  // Public Methods
  public abstract String getChanges();
  public abstract String getDescription();
}
```



B.1.1.12.2. getChanges()

```
public abstract String getChanges();
```

Gets the changes.

Parameters

`return`	the changes

B.1.1.12.3. getDescription()

```
public abstract String getDescription();
```

Gets the description.

Parameters

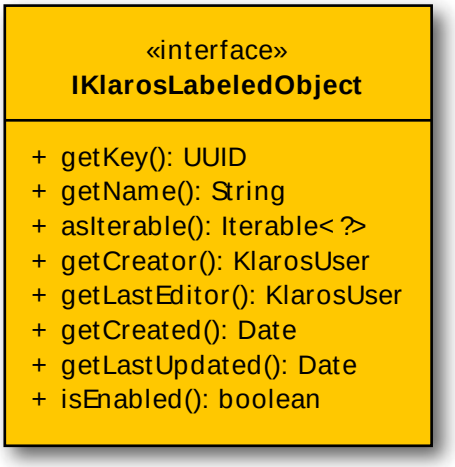
`return`	the description

B.1.1.13. IKlarosLabeledObject

This interface provides access to the fields of a labeled object.

B.1.1.13.1. Synopsis

```
public interface IKlarosLabeledObject {

  // Public Methods
  public abstract Iterable<?> asIterable();
  public abstract Date getCreated();
  public abstract KlarosUser getCreator();
  public abstract UUID getKey();
  public abstract KlarosUser getLastEditor();
  public abstract Date getLastUpdated();
  public abstract String getName();
  public abstract boolean isEnabled();
}
```



B.1.1.13.2. asIterable()

```
public abstract Iterable<?> asIterable();
```

Return this object as an iterable containing just this object.

Parameters

`return`	the iterable container

B.1.1.13.3. getKey()

```
public abstract UUID getKey();
```

Get the internal key of the object. This key is globally unique.

Parameters

`return`	The internal key.

B.1.1.13.4. getName()

```
public abstract String getName();
```

Get the name of the object. This matches the id field visible in the UI.

Parameters

`return`	The name of the object.

B.1.1.13.5. isEnabled()

```
public abstract boolean isEnabled();
```

Returns the value of the enabled flag of this object.

Parameters

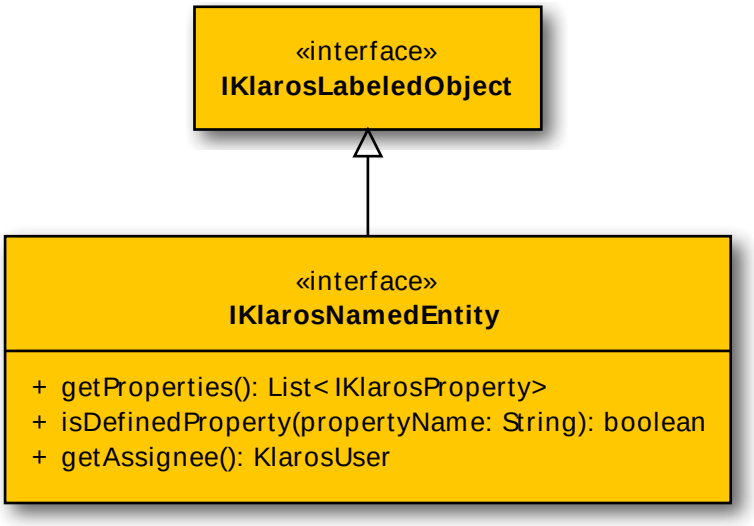
`return`	`true` if the entity is enabled, else `false`

B.1.1.14. IKlarosNamedEntity

This interface provides access to data of a properties owner.

B.1.1.14.1. Synopsis

```
public interface IKlarosNamedEntity extends IKlarosLabeledObject {

  // Public Methods
  public abstract KlarosUser getAssignee();
  public abstract List<IKlarosProperty> getProperties();
  public abstract boolean isDefinedProperty(String propertyName);
}
```



B.1.1.14.2. getProperties()

```
public abstract List<IKlarosProperty> getProperties();
```

Gets the list of properties.

Parameters

`return`	the properties

B.1.1.14.3. isDefinedProperty(String)

```
public abstract boolean isDefinedProperty(String propertyName);
```

Check if a property identified by given name is a defined property.

Parameters

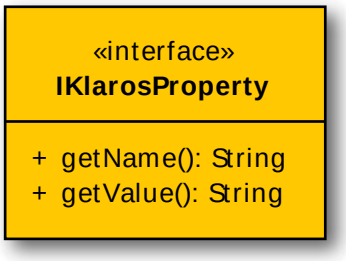
`propertyName`	The name of the property to check.
`return`	`true` if the identified property is a defined property, `false` else.

B.1.1.15. IKlarosProperty

A user defined property.

B.1.1.15.1. Synopsis

```
public interface IKlarosProperty {

  // Public Methods
  public abstract String getName();
  public abstract String getValue();
}
```



B.1.1.15.2. getName()

```
public abstract String getName();
```

Gets the property name.

Parameters

return	the property name

B.1.1.15.3. getValue()

```
public abstract String getValue();
```

Gets the property value.

Parameters

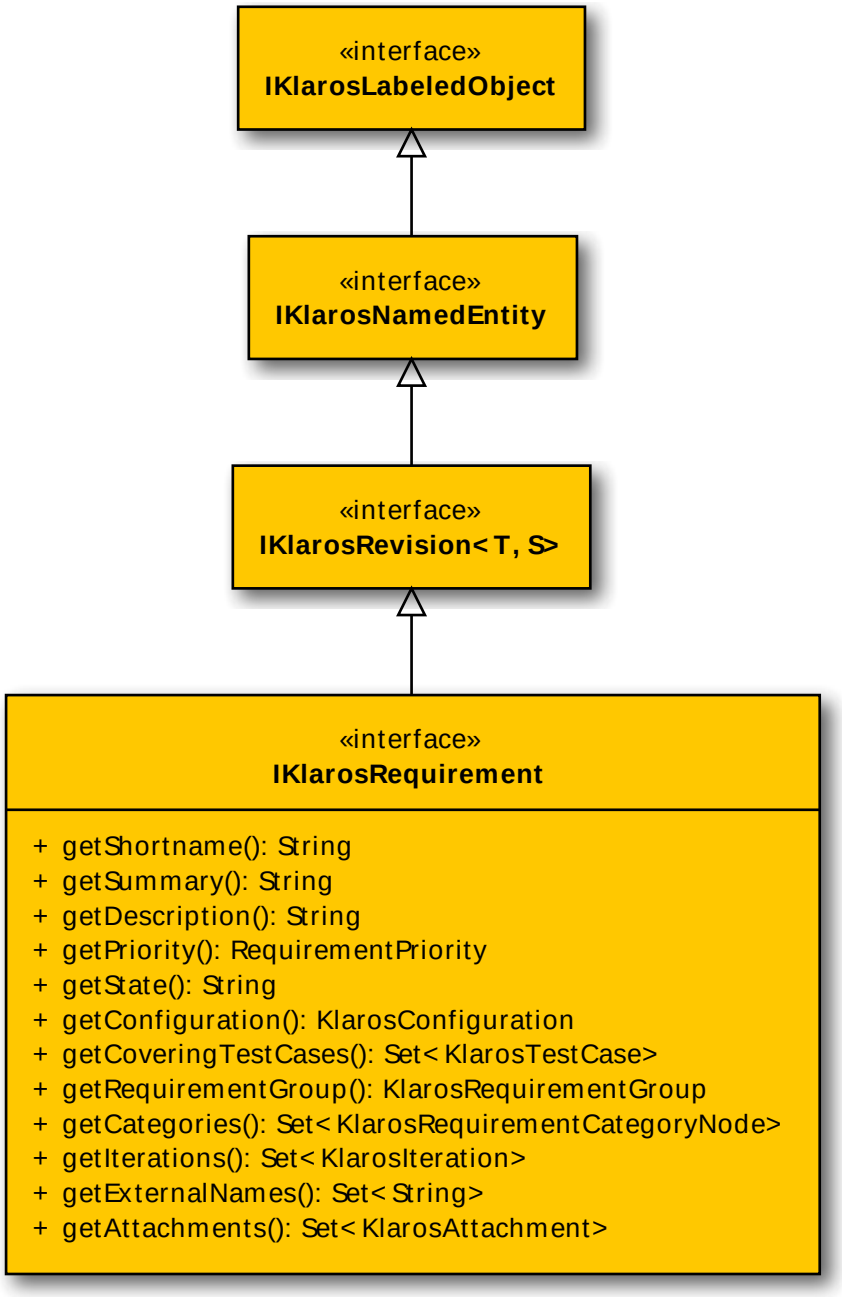
return	the property value

B.1.1.16. IKlarosRequirement

This interface provides access to data of a requirement.

B.1.1.16.1. Synopsis

```
public interface IKlarosRequirement<T,S extends Revision> extends IKlarosRevision<T,
 S> {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosRequirementCategoryNode> getCategories();
  public abstract KlarosConfiguration getConfiguration();
  public abstract Set<KlarosTestCase> getCoveringTestCases();
  public abstract String getDescription();
  public abstract Set<String> getExternalNames();
  public abstract Set<KlarosIteration> getIterations();
  public abstract RequirementPriority getPriority();
  public abstract KlarosRequirementGroup getRequirementGroup();
  public abstract String getShortname();
  public abstract String getState();
  public abstract String getSummary();
}
```



B.1.1.16.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this requirement.

Parameters

`return`	the attachments

B.1.1.16.3. getCategories()

```
public abstract Set<KlarosRequirementCategoryNode> getCategories();
```

Gets the categories this object belongs to. Each category node will belong to a different category tree.

Parameters

`return`	the category nodes

B.1.1.16.4. getConfiguration()

```
public abstract KlarosConfiguration getConfiguration();
```

Get configuration.

Parameters

`return`	The related configuration.

B.1.1.16.5. getCoveringTestCases()

```
public abstract Set<KlarosTestCase> getCoveringTestCases();
```

Get test cases covering the requirement.

Parameters

`return`	`Set` of test cases which cover this test requirement.

B.1.1.16.6. getDescription()

```
public abstract String getDescription();
```

Get the description.

Parameters

`return`	the string

B.1.1.16.7. getExternalNames()

```
public abstract Set<String> getExternalNames();
```

Gets the external requirement names this requirement is associated with. External requirement names are used to match imported requirements to existing requirements.

Parameters

`return`	the external names

B.1.1.16.8. getIterations()

```
public abstract Set<KlarosIteration> getIterations();
```

Get the iterations this requirement is assigned to.

Parameters

`return`	`Set` of iterations this requirement is assigned to.

B.1.1.16.9. getPriority()

```
public abstract RequirementPriority getPriority();
```

Gets the priority.

Parameters

return	the priority

B.1.1.16.10. getRequirementGroup()

```
public abstract KlarosRequirementGroup getRequirementGroup();
```

Get requirement group.

Parameters

return	The requirement group the requirement belongs to.

B.1.1.16.11. getShortname()

```
public abstract String getShortname();
```

Gets the shortname.

Parameters

return	the shortname

B.1.1.16.12. getState()

```
public abstract String getState();
```

Gets the state.

Parameters

return	the state

B.1.1.16.13. getSummary()

```
public abstract String getSummary();
```

Gets the summary.

Parameters

return	the summary

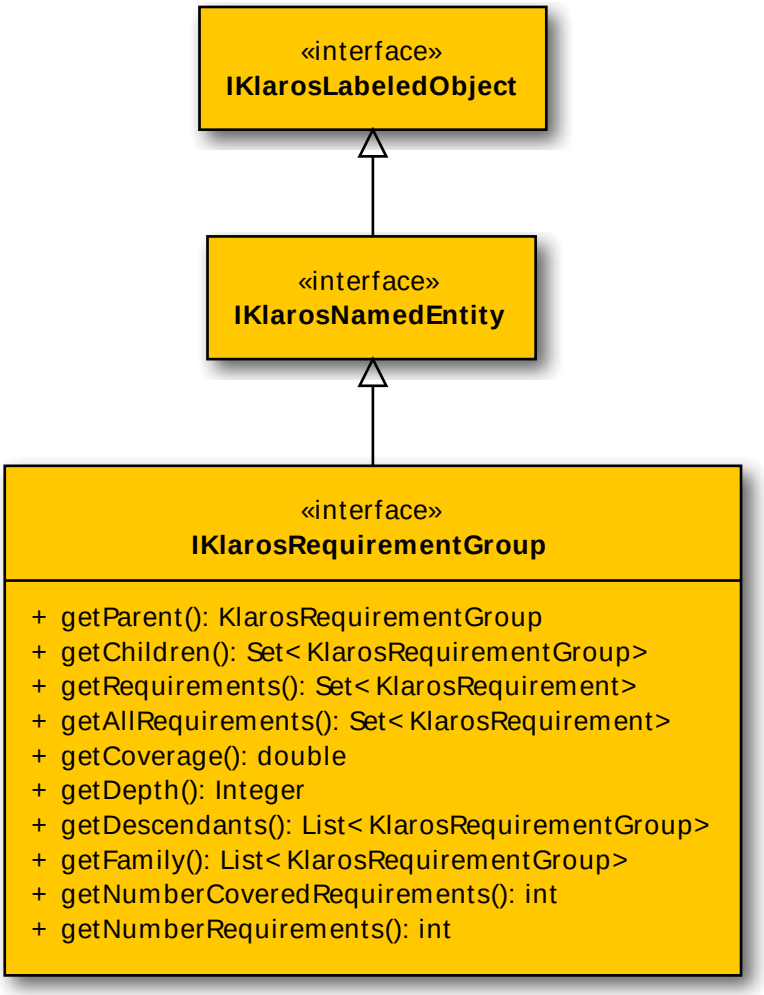
B.1.1.17. IKlarosRequirementGroup

The interface to retrieve the information about a requirement group.

B.1.1.17.1. Synopsis

```
public interface IKlarosRequirementGroup extends IKlarosNamedEntity {
```

```
  // Public Methods
  public abstract Set<KlarosRequirement> getAllRequirements();
  public abstract Set<KlarosRequirementGroup> getChildren();
  public abstract double getCoverage();
  public abstract Integer getDepth();
  public abstract List<KlarosRequirementGroup> getDescendants();
  public abstract List<KlarosRequirementGroup> getFamily();
  public abstract int getNumberCoveredRequirements();
  public abstract int getNumberRequirements();
  public abstract KlarosRequirementGroup getParent();
  public abstract Set<KlarosRequirement> getRequirements();
}
```



B.1.1.17.2. getAllRequirements()

```
  public abstract Set<KlarosRequirement> getAllRequirements();
```

Get all requirements of this group hierarchy.

Parameters

`return`	`List` containing own requirements and all requirements of groups which are descendants of this group.

B.1.1.17.3. getChildren()

```
public abstract Set<KlarosRequirementGroup> getChildren();
```

Get the children of this requirement group.

Parameters

`return`	`Set` containing the requirement groups that are children of this requirement group.

B.1.1.17.4. getCoverage()

```
public abstract double getCoverage();
```

Get test requirement coverage for the group.

Parameters

`return`	the coverage in percent. If no requirements are found, `1.0` is returned.

B.1.1.17.5. getDepth()

```
public abstract Integer getDepth();
```

Get depth in tree of this node.

Parameters

`return`	The depth in the tree

B.1.1.17.6. getDescendants()

```
public abstract List<KlarosRequirementGroup> getDescendants();
```

Get all descending subgroups of this group.

Parameters

`return`	`List` containing all groups which are descendants of this group.

B.1.1.17.7. getFamily()

```
public abstract List<KlarosRequirementGroup> getFamily();
```

Get whole family with this group as root.

Parameters

`return`	List containing this group and all of its descendants.

B.1.1.17.8. getNumberCoveredRequirements()

```
public abstract int getNumberCoveredRequirements();
```

Get number of test requirements covered by at least one test case.

Parameters

`return`	The number of covered test requirements.

B.1.1.17.9. getNumberRequirements()

```
public abstract int getNumberRequirements();
```

Get number of test requirements, including all descendants' test requirements.

Parameters

`return`	The number of overall test requirements.

B.1.1.17.10. getParent()

```
public abstract KlarosRequirementGroup getParent();
```

Get the parent of this requirement group.

Parameters

`return`	The parent requirement group, or `null`.

B.1.1.17.11. getRequirements()

```
public abstract Set<KlarosRequirement> getRequirements();
```

Get the related requirements of this requirement group.

Parameters

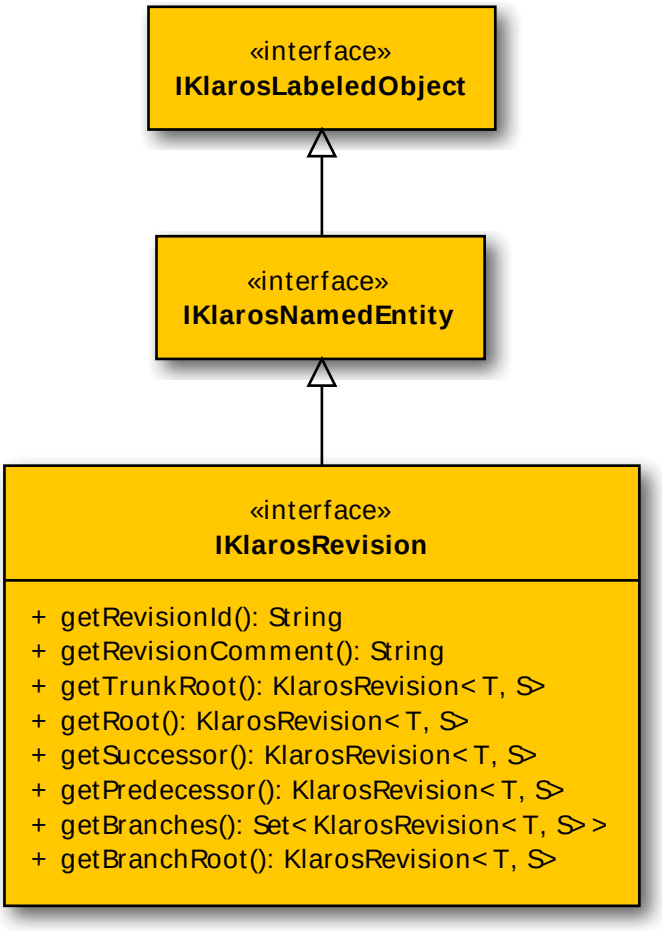
`return`	`Set` containing the requirements which belong to this requirement group.

B.1.1.18. IKlarosRevision

This interface provides access to a revisionable Klaros object.

B.1.1.18.1. Synopsis

```
public interface IKlarosRevision<T,S extends Revision> extends IKlarosNamedEntity {

  // Public Methods
  public abstract KlarosRevision<T, S> getBranchRoot();
  public abstract Set<KlarosRevision<T, S>> getBranches();
  public abstract KlarosRevision<T, S> getPredecessor();
  public abstract String getRevisionComment();
  public abstract String getRevisionId();
  public abstract KlarosRevision<T, S> getRoot();
  public abstract KlarosRevision<T, S> getSuccessor();
  public abstract KlarosRevision<T, S> getTrunkRoot();
}
```



B.1.1.18.2. getBranches()

```
public abstract Set<KlarosRevision<T, S>> getBranches();
```

Get branches of this revision.

Parameters

`return`	`Set` containing all revision objects which have been created as first revision of a branch (trunk root) and habe been created using this revision.

B.1.1.18.3. getBranchRoot()

```
public abstract KlarosRevision<T, S> getBranchRoot();
```

Get the original revision of a trunk root revision object.

Parameters

`return`	The revision object that has been used to create a new branch.

B.1.1.18.4. getPredecessor()

```
public abstract KlarosRevision<T, S> getPredecessor();
```

Get the predecessor of the revision.

Parameters

`return`	The revision object that is the predecessor of this revision.

B.1.1.18.5. getRevisionComment()

```
public abstract String getRevisionComment();
```

Get comment.

Parameters

`return`	The comment of the revision.

B.1.1.18.6. getRevisionId()

```
public abstract String getRevisionId();
```

Get the revision id.

Parameters

`return`	The revision id.

B.1.1.18.7. getRoot()

```
public abstract KlarosRevision<T, S> getRoot();
```

Get the root of the revision hierarchy.

Parameters

`return`	The root revision object.

B.1.1.18.8. getSuccessor()

```
public abstract KlarosRevision<T, S> getSuccessor();
```

Get the successor of the revision.

Parameters

`return`	The revision object that is the successor of this revision.

B.1.1.18.9. getTrunkRoot()

```
public abstract KlarosRevision<T, S> getTrunkRoot();
```

Get the 'root' of a trunk.

Parameters

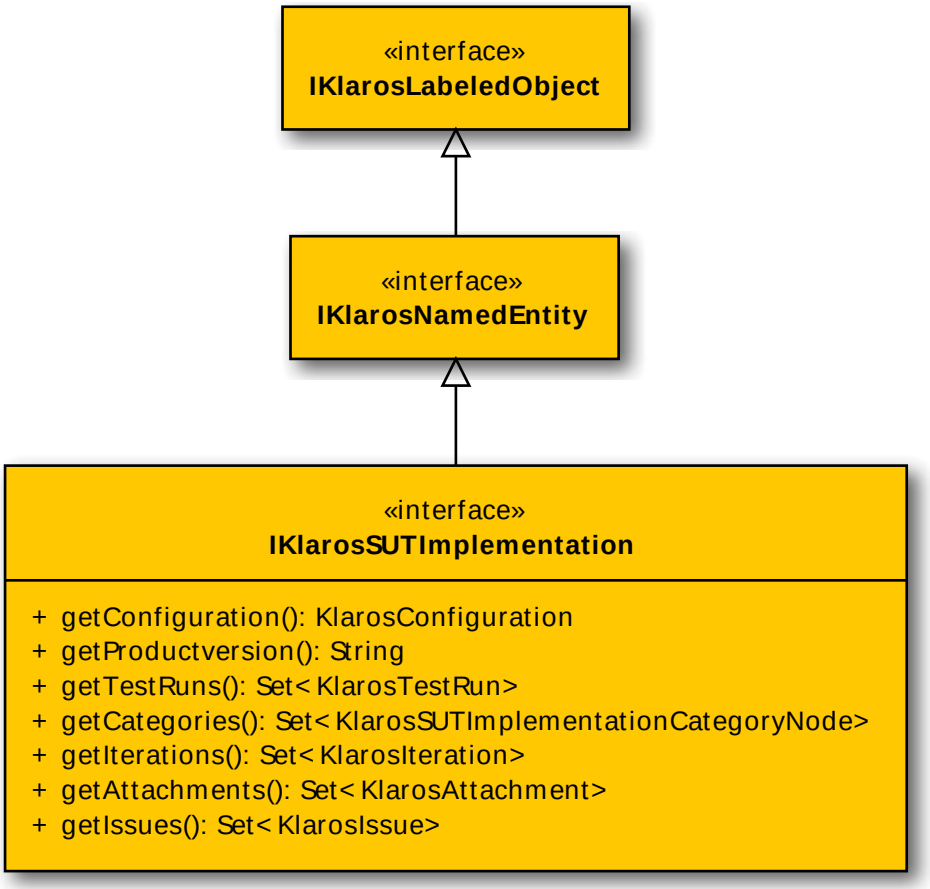
`return`	the revision which is the first revision of the branch to which this revision belongs.

B.1.1.19. IKlarosSUTImplementation

This interface provides access to data of a system under test version.

B.1.1.19.1. Synopsis

```
public interface IKlarosSUTImplementation extends IKlarosNamedEntity {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosSUTImplementationCategoryNode> getCategories();
  public abstract KlarosConfiguration getConfiguration();
  public abstract Set<KlarosIssue> getIssues();
  public abstract Set<KlarosIteration> getIterations();
  public abstract String getProductversion();
  public abstract Set<KlarosTestRun> getTestRuns();
}
```



B.1.1.19.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this system under test.

Parameters

`return`	the attachments

B.1.1.19.3. getCategories()

```
public abstract Set<KlarosSUTImplementationCategoryNode> getCategories();
```

Gets the categories this system under test belongs to. Each category node will belong to a different category tree.

Parameters

`return`	the category nodes

B.1.1.19.4. getConfiguration()

```
public abstract KlarosConfiguration getConfiguration();
```

Get configuration.

Parameters

`return`	The related configuration.

B.1.1.19.5. getIssues()

```
public abstract Set<KlarosIssue> getIssues();
```

Gets the issues related to this system under test.

Parameters

`return`	the issues

B.1.1.19.6. getIterations()

```
public abstract Set<KlarosIteration> getIterations();
```

Get the iterations this system under test is assigned to.

Parameters

`return`	`Set` of iterations this system under test is assigned to.

B.1.1.19.7. getProductversion()

```
public abstract String getProductversion();
```

Get product version.

Parameters

`return`	The version id of the system under test.

B.1.1.19.8. getTestRuns()

```
public abstract Set<KlarosTestRun> getTestRuns();
```

Get the test runs performed with this system under test.

Parameters

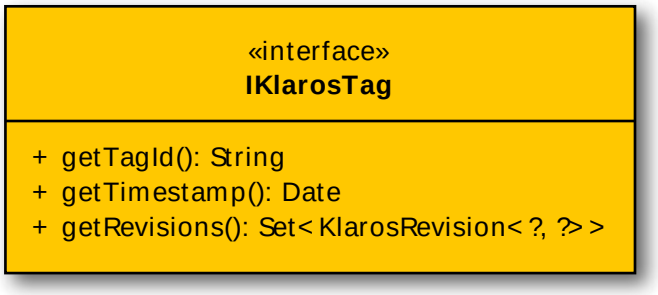
`return`	`Set` of test runs performed for this system under test.

B.1.1.20. IKlarosTag

This interface provides access to the data of a tag.

B.1.1.20.1. Synopsis

```
public interface IKlarosTag {

  // Public Methods
  public abstract Set<KlarosRevision<?, ?>> getRevisions();
  public abstract String getTagId();
  public abstract Date getTimestamp();
}
```



B.1.1.20.2. getRevisions()

```
public abstract Set<KlarosRevision<?, ?>> getRevisions();
```

Get related revisions.

Parameters

`return`	`Collection` of revision objects which are related to this tag.

B.1.1.20.3. getTagId()

```
public abstract String getTagId();
```

Get tag id.

Parameters

`return`	The id of the tag.

B.1.1.20.4. getTimestamp()

```
public abstract Date getTimestamp();
```

Get timestamp.

Parameters

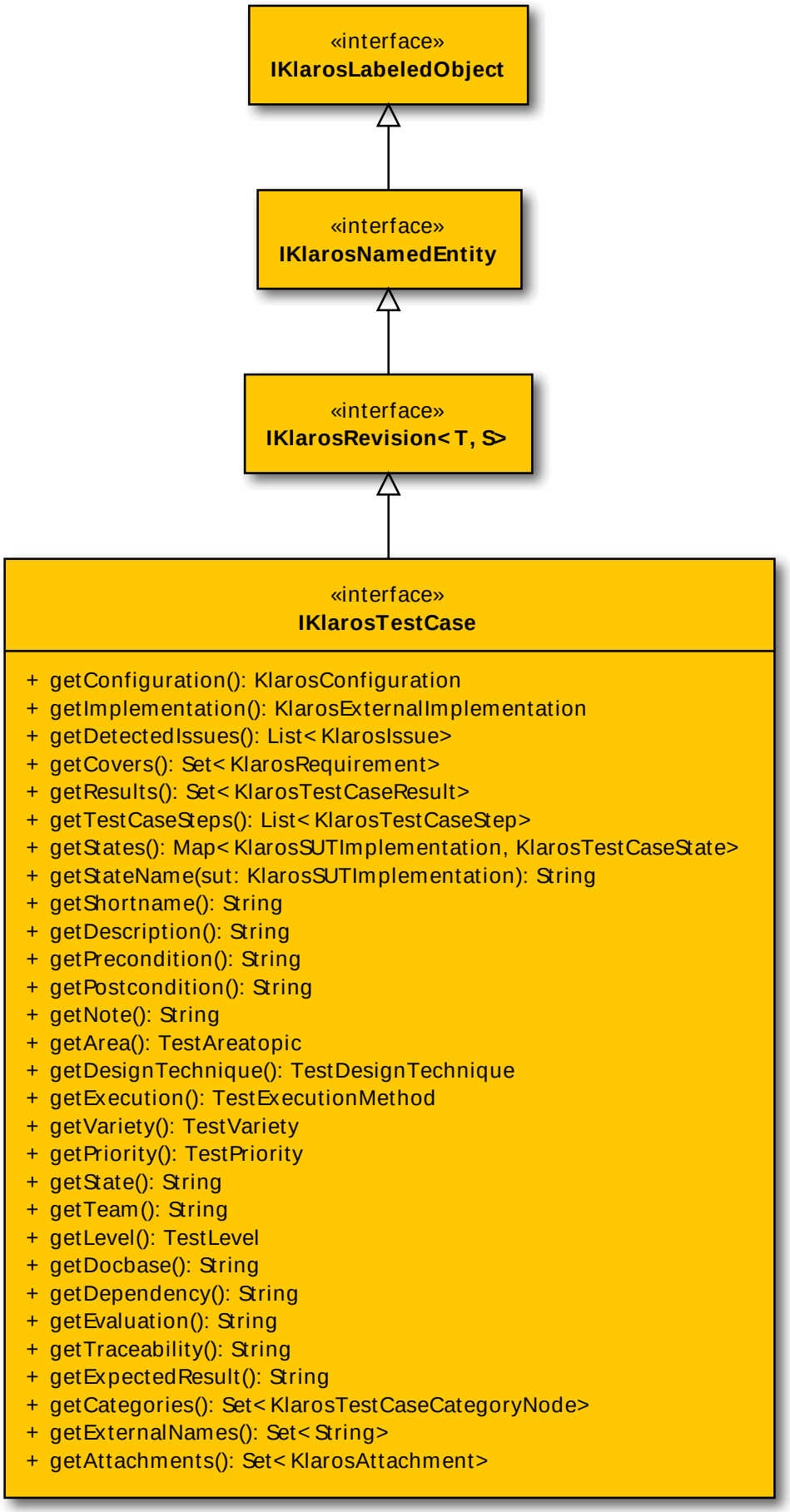
`return`	The time of the creation of the tag as a `Date` object.

B.1.1.21. IKlarosTestCase

This class provides access to data of a test case.

B.1.1.21.1. Synopsis

```
public interface IKlarosTestCase<T,S extends Revision> extends IKlarosRevision<T,
 S> {

  // Public Methods
  public abstract TestAreatopic getArea();
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosTestCaseCategoryNode> getCategories();
  public abstract KlarosConfiguration getConfiguration();
  public abstract Set<KlarosRequirement> getCovers();
  public abstract String getDependency();
  public abstract String getDescription();
  public abstract TestDesignTechnique getDesignTechnique();
  public abstract List<KlarosIssue> getDetectedIssues();
  public abstract String getDocbase();
  public abstract String getEvaluation();
  public abstract TestExecutionMethod getExecution();
  public abstract String getExpectedResult();
  public abstract Set<String> getExternalNames();
  public abstract KlarosExternalImplementation getImplementation();
  public abstract TestLevel getLevel();
  public abstract String getNote();
  public abstract String getPostcondition();
  public abstract String getPrecondition();
  public abstract TestPriority getPriority();
  public abstract Set<KlarosTestCaseResult> getResults();
  public abstract String getShortname();
  public abstract String getState();
  public abstract String getStateName(KlarosSUTImplementation sut);
  public abstract Map<KlarosSUTImplementation, KlarosTestCaseState> getStates();
  public abstract String getTeam();
  public abstract List<KlarosTestCaseStep> getTestCaseSteps();
  public abstract String getTraceability();
  public abstract TestVariety getVariety();
}
```


«interface»
IKlarosLabeledObject
«interface»
IKlarosNamedEntity
«interface»
IKlarosRevision< T, S>
«interface»
IKlarosTestCase
+ getConfiguration(): KlarosConfiguration
+ getImplementation(): KlarosExternalImplementation
+ getDetectedIssues(): List< KlarosIssue>
+ getCovers(): Set< KlarosRequirement>
+ getResults(): Set< KlarosTestCaseResult>
+ getTestCaseSteps(): List< KlarosTestCaseStep>
+ getStates(): Map< KlarosSUTImplementation, KlarosTestCaseState>
+ getStateName(sut: KlarosSUTImplementation): String
+ getShortname(): String
+ getDescription(): String
+ getPrecondition(): String
+ getPostcondition(): String
+ getNote(): String
+ getArea(): TestAreatopic
+ getDesignTechnique(): TestDesignTechnique
+ getExecution(): TestExecutionMethod
+ getVariety(): TestVariety
+ getPriority(): TestPriority
+ getState(): String
+ getTeam(): String
+ getLevel(): TestLevel
+ getDocbase(): String
+ getDependency(): String
+ getEvaluation(): String
+ getTraceability(): String
+ getExpectedResult(): String
+ getCategories(): Set< KlarosTestCaseCategoryNode>
+ getExternalNames(): Set< String>
+ getAttachments(): Set< KlarosAttachment>

B.1.1.21.2. getArea()

```
public abstract TestAreatopic getArea();
```

The area of this test case.

Parameters

`return`	The area.

B.1.1.21.3. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this test case.

Parameters

`return`	the attachments

B.1.1.21.4. getCategories()

```
public abstract Set<KlarosTestCaseCategoryNode> getCategories();
```

Gets the categories this object belongs to. Each category node will belong to a different category tree.

Parameters

`return`	the category nodes

B.1.1.21.5. getConfiguration()

```
public abstract KlarosConfiguration getConfiguration();
```

Get the project configuration this test case revision belongs to.

Parameters

`return`	The related configuration.

B.1.1.21.6. getCovers()

```
public abstract Set<KlarosRequirement> getCovers();
```

Get covered requirements.

Parameters

`return`	`Set` of requirements which are covered by this test case revision.

B.1.1.21.7. getDependency()

```
public abstract String getDependency();
```

The dependency of this test case.

Parameters

`return`	The dependency.

B.1.1.21.8. getDescription()

```
public abstract String getDescription();
```

The description of this test case.

Parameters

`return`	The description.

B.1.1.21.9. getDesignTechnique()

```
public abstract TestDesignTechnique getDesignTechnique();
```

The design technique of this test case.

Parameters

`return`	The type.

B.1.1.21.10. getDetectedIssues()

```
public abstract List<KlarosIssue> getDetectedIssues();
```

Get detected issues.

Parameters

`return`	`List` of issues which have been detected by this test case revision.

B.1.1.21.11. getDocbase()

```
public abstract String getDocbase();
```

The docbase of this test case.

Parameters

`return`	The docbase.

B.1.1.21.12. getEvaluation()

```
public abstract String getEvaluation();
```

The evaluation of this test case.

Parameters

`return`	The evaluation.

B.1.1.21.13. getExecution()

```
public abstract TestExecutionMethod getExecution();
```

The execution method of this test case.

Parameters

`return`	The execution method.

B.1.1.21.14. getExpectedResult()

```
public abstract String getExpectedResult();
```

The expected result of this test case.

Parameters

`return`	The expected result.

B.1.1.21.15. getExternalNames()

```
public abstract Set<String> getExternalNames();
```

Gets the external test case names this test case is associated with. External test case names are used to match imported test case result to existing test cases.

Parameters

`return`	the external names

B.1.1.21.16. getImplementation()

```
public abstract KlarosExternalImplementation getImplementation();
```

Get implementation.

Parameters

`return`	The object describing the location of the implementation of this test case revision.

B.1.1.21.17. getLevel()

```
public abstract TestLevel getLevel();
```

The level of this test case.

Parameters

`return`	The level.

B.1.1.21.18. getNote()

```
public abstract String getNote();
```

The note of this test case.

Parameters

`return`	The note.

B.1.1.21.19. getPostcondition()

```
public abstract String getPostcondition();
```

The postcondition of this test case.

Parameters

`return`	The postcondition.

B.1.1.21.20. getPrecondition()

```
public abstract String getPrecondition();
```

The precondition of this test case.

Parameters

`return`	The precondition.

B.1.1.21.21. getPriority()

```
public abstract TestPriority getPriority();
```

The priority of this test case.

Parameters

`return`	The priority.

B.1.1.21.22. getResults()

```
public abstract Set<KlarosTestCaseResult> getResults();
```

Get test case results.

Parameters

`return`	`Set` of results of executions of this test case revision.

B.1.1.21.23. getShortname()

```
public abstract String getShortname();
```

The short name (title) of this test case.

Parameters

`return`	The short name.

B.1.1.21.24. getState()

```
public abstract String getState();
```

The type of this test case.

Parameters

`return`	The type.

B.1.1.21.25. getStateName(KlarosSUTImplementation)

```
public abstract String getStateName(KlarosSUTImplementation sut);
```

Get name of the state related to given SUT.

Parameters

`sut`	The related SUT implementation
`return`	The name of the state

B.1.1.21.26. getStates()

```
public abstract Map<KlarosSUTImplementation, KlarosTestCaseState> getStates();
```

Get test case states.

Parameters

`return`	`Collection` of the states of this test case revision in relation to the different SUT versions.-.

B.1.1.21.27. getTeam()

```
public abstract String getTeam();
```

The team responsible for this test case.

Parameters

`return`	The team.

B.1.1.21.28. getTestCaseSteps()

```
public abstract List<KlarosTestCaseStep> getTestCaseSteps();
```

Get test case steps.

Parameters

`return`	`list` of steps of this test case revision.

B.1.1.21.29. getTraceability()

```
public abstract String getTraceability();
```

The traceability of this test case.

Parameters

`return`	The traceability.

B.1.1.21.30. getVariety()

```
public abstract TestVariety getVariety();
```

The variety of this test case.

Parameters

`return`	The variety.

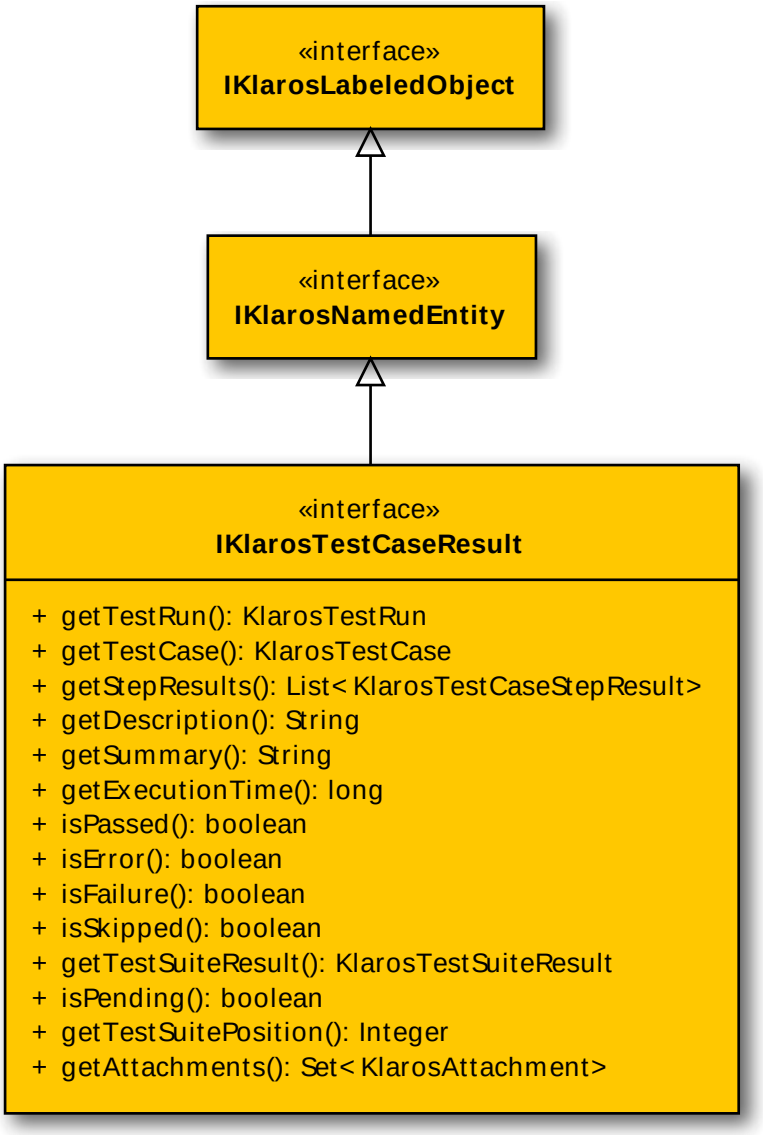
B.1.1.22. IKlarosTestCaseResult

This interface provides access to data of a test case result.

B.1.1.22.1. Synopsis

```
public interface IKlarosTestCaseResult extends IKlarosNamedEntity {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
```

```
    public abstract String getDescription();
    public abstract long getExecutionTime();
    public abstract List<KlarosTestCaseStepResult> getStepResults();
    public abstract String getSummary();
    public abstract KlarosTestCase getTestCase();
    public abstract KlarosTestRun getTestRun();
    public abstract Integer getTestSuitePosition();
    public abstract KlarosTestSuiteResult getTestSuiteResult();
    public abstract boolean isError();
    public abstract boolean isFailure();
    public abstract boolean isPassed();
    public abstract boolean isPending();
    public abstract boolean isSkipped();
}
```



B.1.1.22.2. getAttachments()

```
    public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this test case result.

Parameters

`return`	the attachments

B.1.1.22.3. getDescription()

```
public abstract String getDescription();
```

Get the test result description. This is usually set for failed/error status results.

Parameters

`return`	The test result description.

B.1.1.22.4. getExecutionTime()

```
public abstract long getExecutionTime();
```

Get the test execution time in ms.

Parameters

`return`	The test execution time.

B.1.1.22.5. getStepResults()

```
public abstract List<KlarosTestCaseStepResult> getStepResults();
```

Get the associated step results.

Parameters

`return`	`List` of test case step results.

B.1.1.22.6. getSummary()

```
public abstract String getSummary();
```

Get the test result summary. This is usually set for failed/error status results.

Parameters

`return`	The test result summary.

B.1.1.22.7. getTestCase()

```
public abstract KlarosTestCase getTestCase();
```

Get the associated test case.

Parameters

`return`	The test case that has been executed to get this result.

B.1.1.22.8. getTestRun()

```
public abstract KlarosTestRun getTestRun();
```

Get the associated test run.

Parameters

`return`	The test run that created this result.

B.1.1.22.9. getTestSuitePosition()

```
public abstract Integer getTestSuitePosition();
```

Return the position in the the test suite result this test case result belongs to or null if this test case has not been executed by executing a test suite.

Parameters

`return`	the position of the test case in the test suite, or null

B.1.1.22.10. getTestSuiteResult()

```
public abstract KlarosTestSuiteResult getTestSuiteResult();
```

Return the test suite result this test case result belongs to or null if this test case has not been executed by executing a test suite.

Parameters

`return`	the test suite result, or null

B.1.1.22.11. isError()

```
public abstract boolean isError();
```

Check if this is an error result. It is assumed, that error results have a property 'type' with the value 'E' or 'error'.

Parameters

`return`	`true` if this results represents an error.

B.1.1.22.12. isFailure()

```
public abstract boolean isFailure();
```

Check if this is a failure result. It is assumed, that failure results have a property 'type' with the value 'F' or 'failure'.

Parameters

`return`	`true` if this results represents a failure.

B.1.1.22.13. isPassed()

```
public abstract boolean isPassed();
```

Check if this is a result of a passed test case. It is assumed, that passed results have a property 'testCasePassed' with value 'true'.

Parameters

`return`	`true` if this results represents an error.

B.1.1.22.14. isPending()

```
public abstract boolean isPending();
```

Checks if the test case result is pending. This means that it has not yet been completely executed.

Parameters

`return`	true, if the test case result is pending

B.1.1.22.15. isSkipped()

```
public abstract boolean isSkipped();
```

Check if this is a skipped result. It is assumed, that skipped results have a property 'type' with the value 'S' or 'skipped'.

Parameters

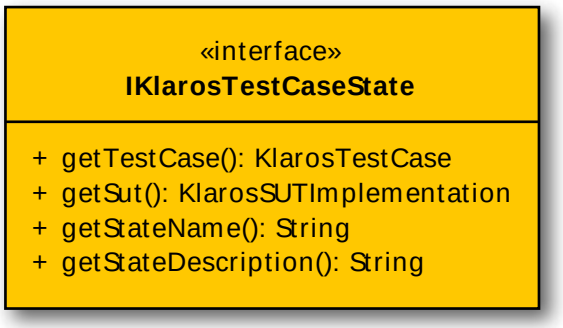
`return`	`true` if this results represents a skipped result.

B.1.1.23. IKlarosTestCaseState

This interface provides access to data of a test case state.

B.1.1.23.1. Synopsis

```
public interface IKlarosTestCaseState {

  // Public Methods
  public abstract String getStateDescription();
  public abstract String getStateName();
  public abstract KlarosSUTImplementation getSut();
  public abstract KlarosTestCase getTestCase();
}
```



B.1.1.23.2. getStateDescription()

```
public abstract String getStateDescription();
```

Get description of state.

Parameters

`return`	The description of this state.

B.1.1.23.3. getStateName()

```
public abstract String getStateName();
```

Get name of state.

Parameters

return	The name of this state.

B.1.1.23.4. getSut()

```
public abstract KlarosSUTImplementation getSut();
```

Get related system under test version.

Parameters

return	The related SUT version.

B.1.1.23.5. getTestCase()

```
public abstract KlarosTestCase getTestCase();
```

Get test case.

Parameters

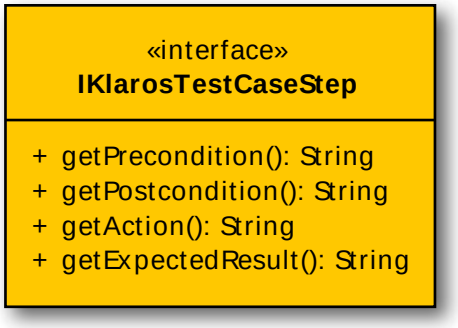
return	The test case whose state is defined by this state object.

B.1.1.24. IKlarosTestCaseStep

This interface provides access to a test case step.

B.1.1.24.1. Synopsis

```
public interface IKlarosTestCaseStep {

  // Public Methods
  public abstract String getAction();
  public abstract String getExpectedResult();
  public abstract String getPostcondition();
  public abstract String getPrecondition();
}
```



B.1.1.24.2. getAction()

```
public abstract String getAction();
```

Get the action.

Parameters

return	The action of this test case step.

B.1.1.24.3. getExpectedResult()

```
public abstract String getExpectedResult();
```

Get the expected result.

Parameters

return	The expected result of this test case step.

B.1.1.24.4. getPostcondition()

```
public abstract String getPostcondition();
```

Get the postcondition.

Parameters

return	The postcondition of this test case step.

B.1.1.24.5. getPrecondition()

```
public abstract String getPrecondition();
```

Get the precondition.

Parameters

return	The precondition of this test case step.

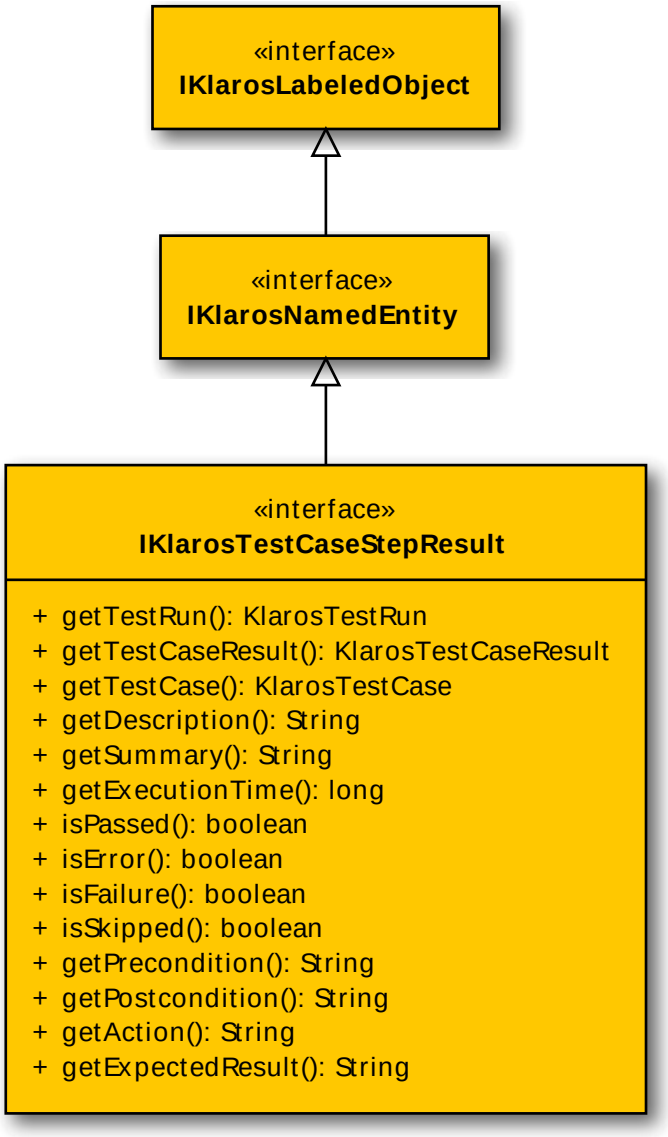
B.1.1.25. IKlarosTestCaseStepResult

This interface provides access to data of a test case step result.

B.1.1.25.1. Synopsis

```
public interface IKlarosTestCaseStepResult extends IKlarosNamedEntity {

  // Public Methods
  public abstract String getAction();
  public abstract String getDescription();
  public abstract long getExecutionTime();
  public abstract String getExpectedResult();
  public abstract String getPostcondition();
  public abstract String getPrecondition();
  public abstract String getSummary();
  public abstract KlarosTestCase getTestCase();
  public abstract KlarosTestCaseResult getTestCaseResult();
```

```
  public abstract KlarosTestRun getTestRun();
  public abstract boolean isError();
  public abstract boolean isFailure();
  public abstract boolean isPassed();
  public abstract boolean isSkipped();
}
```



B.1.1.25.2. getAction()

```
  public abstract String getAction();
```

Get the action.

Parameters

`return`	The action of this test case step.

B.1.1.25.3. getDescription()

```
  public abstract String getDescription();
```

Get the test result description. This is usually set for failed/error status results.

Parameters

`return`	The test result description.

B.1.1.25.4. getExecutionTime()

```
public abstract long getExecutionTime();
```

Get the test execution time in ms.

Parameters

`return`	The test execution time.

B.1.1.25.5. getExpectedResult()

```
public abstract String getExpectedResult();
```

Get the expected result.

Parameters

`return`	The expected result of this test case step.

B.1.1.25.6. getPostcondition()

```
public abstract String getPostcondition();
```

Get the postcondition.

Parameters

`return`	The postcondition of this test case step.

B.1.1.25.7. getPrecondition()

```
public abstract String getPrecondition();
```

Get the precondition.

Parameters

`return`	The precondition of this test case step.

B.1.1.25.8. getSummary()

```
public abstract String getSummary();
```

Get the test result summary. This is usually set for failed/error status results.

Parameters

`return`	The test result summary.

B.1.1.25.9. getTestCase()

```
public abstract KlarosTestCase getTestCase();
```

Get test case.

Parameters

`return`	The test case that has been executed to get this result.

B.1.1.25.10. getTestCaseResult()

```
public abstract KlarosTestCaseResult getTestCaseResult();
```

Get test case result.

Parameters

`return`	The test case result that has been executed to get this result.

B.1.1.25.11. getTestRun()

```
public abstract KlarosTestRun getTestRun();
```

Get test run.

Parameters

`return`	The test run that created this result.

B.1.1.25.12. isError()

```
public abstract boolean isError();
```

Check if this is an error result. It is assumed, that error results have a property 'type' with the value 'E' or 'error'.

Parameters

`return`	`true` if this results represents an error.

B.1.1.25.13. isFailure()

```
public abstract boolean isFailure();
```

Check if this is a failure result. It is assumed, that failure results have a property 'type' with the value 'F' or 'failure'.

Parameters

`return`	`true` if this results represents a failure.

B.1.1.25.14. isPassed()

```
public abstract boolean isPassed();
```

Check if this is a result of a passed test case. It is assumed, that passed results have a property 'testCasePassed' with value 'true'.

Parameters

`return`	`true` if this results represents an error.

B.1.1.25.15. isSkipped()

```
public abstract boolean isSkipped();
```

Check if this is a skipped result. It is assumed, that skipped results have a property 'type' with the value 'S' or 'skipped'.

Parameters

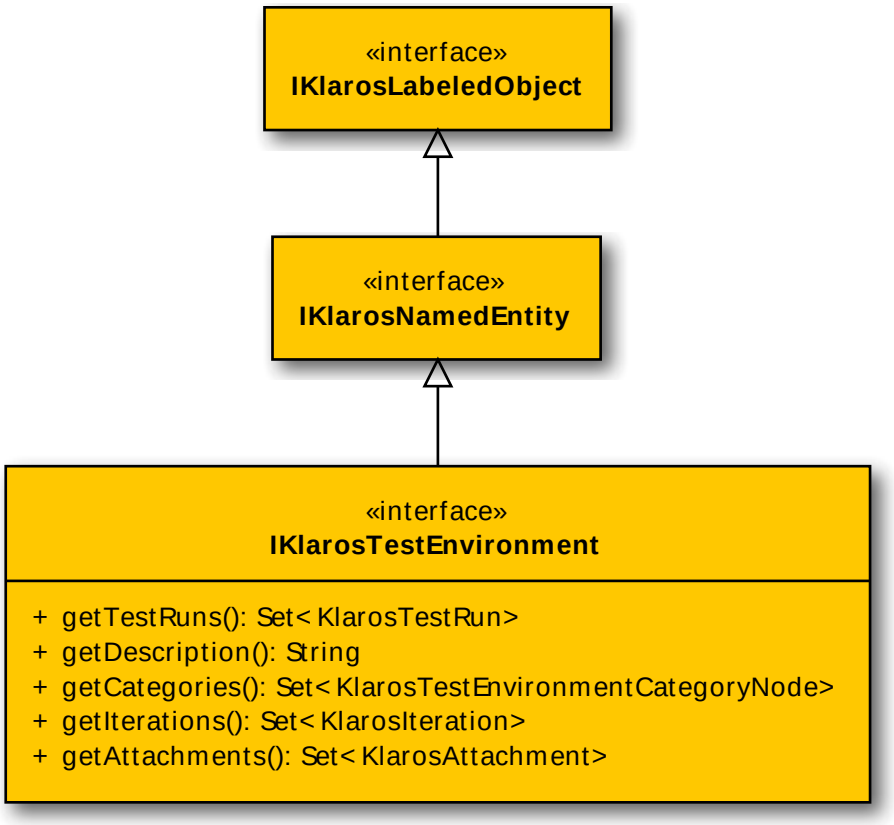
`return` `true` if this results represents a skipped result.

B.1.1.26. IKlarosTestEnvironment

This interface provides access to data of a test environment.

B.1.1.26.1. Synopsis

```
public interface IKlarosTestEnvironment extends IKlarosNamedEntity {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosTestEnvironmentCategoryNode> getCategories();
  public abstract String getDescription();
  public abstract Set<KlarosIteration> getIterations();
  public abstract Set<KlarosTestRun> getTestRuns();
}
```



B.1.1.26.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this test environment.

Parameters

`return` the attachments

B.1.1.26.3. getCategories()

```
public abstract Set<KlarosTestEnvironmentCategoryNode> getCategories();
```

Gets the categories this test environment belongs to. Each category node will belong to a different category tree.

Parameters

`return` the category nodes

B.1.1.26.4. getIterations()

```
public abstract Set<KlarosIteration> getIterations();
```

Get the iterations this test environment is assigned to.

Parameters

`return` `Set` of iterations this test environment is assigned to.

B.1.1.26.5. getTestRuns()

```
public abstract Set<KlarosTestRun> getTestRuns();
```

Gets the test runs associated with this test environment.

Parameters

`return` `Collection` of test runs executed in the environment.

B.1.1.27. IKlarosTestExecutable

This class provides access to a test executable. This may wrap a test case or a test suite.

B.1.1.27.1. Synopsis

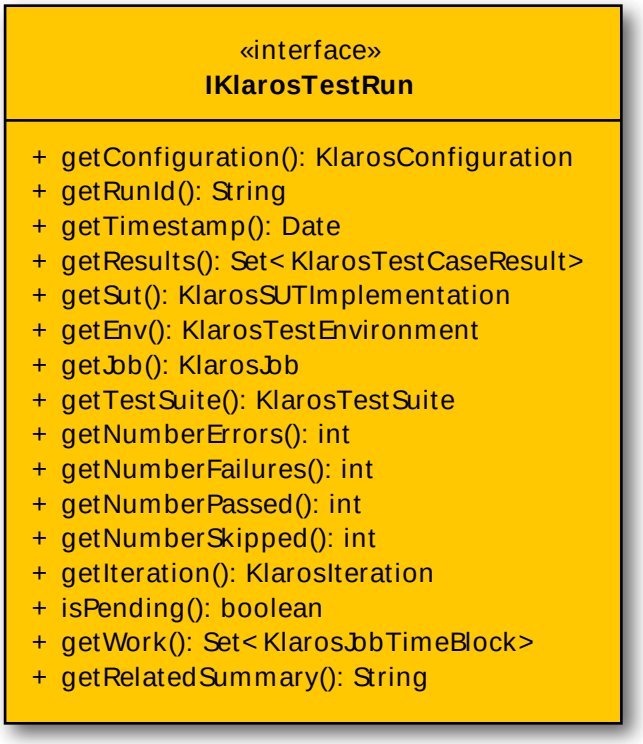
```
public interface IKlarosTestExecutable {
}
```

B.1.1.28. IKlarosTestRun

This interface provides access to data of a test run.

B.1.1.28.1. Synopsis

```
public interface IKlarosTestRun {

  // Public Methods
  public abstract KlarosConfiguration getConfiguration();
  public abstract KlarosTestEnvironment getEnv();
  public abstract KlarosIteration getIteration();
  public abstract KlarosJob getJob();
  public abstract int getNumberErrors();
  public abstract int getNumberFailures();
  public abstract int getNumberPassed();
  public abstract int getNumberSkipped();
  public abstract String getRelatedSummary();
  public abstract Set<KlarosTestCaseResult> getResults();
  public abstract String getRunId();
  public abstract KlarosSUTImplementation getSut();
  public abstract KlarosTestSuite getTestSuite();
  public abstract Date getTimestamp();
  public abstract Set<KlarosJobTimeBlock> getWork();
  public abstract boolean isPending();
}
```



B.1.1.28.2. getConfiguration()

```
public abstract KlarosConfiguration getConfiguration();
```

Get configuration.

Parameters

`return`	The related configuration

B.1.1.28.3. getEnv()

```
public abstract KlarosTestEnvironment getEnv();
```

Get test environment.

Parameters

`return`	The test environment in which the test cases have been executed.

B.1.1.28.4. getIteration()

```
public abstract KlarosIteration getIteration();
```

Get the iteration in which the test run has been created, if available.

Parameters

`return`	The iteration in which the test run has been created

B.1.1.28.5. getJob()

```
public abstract KlarosJob getJob();
```

Get the job that initiated the test run, if available.

Parameters

`return`	The job that initiated the test run

B.1.1.28.6. getNumberErrors()

```
public abstract int getNumberErrors();
```

Get number of test cases with errors of this test run. It is assumed, that error results have a property 'type' with the value 'E'.

Parameters

`return`	The number of error test cases

B.1.1.28.7. getNumberFailures()

```
public abstract int getNumberFailures();
```

Get number of failed test cases of this test run. It is assumed, that failed results have a property 'type' with the value 'F'.

Parameters

`return`	The number of failed test cases

B.1.1.28.8. getNumberPassed()

```
public abstract int getNumberPassed();
```

Get number of passed test cases of this test run. It is assumed, that passed results have a property 'testCasePassed' with value 'true'.

Parameters

`return`	The number of passed test cases

B.1.1.28.9. getNumberSkipped()

```
public abstract int getNumberSkipped();
```

Get number of skipped test cases of this test run. It is assumed, that failed results have a property 'type' with the value 'S'.

Parameters

`return`	The number of skipped test cases

B.1.1.28.10. getRelatedSummary()

```
public abstract String getRelatedSummary();
```

Gets the summary of the job, test suite or test case related to this test run.

Parameters

`return`	the related summary

B.1.1.28.11. getResults()

```
public abstract Set<KlarosTestCaseResult> getResults();
```

Get results.

Parameters

`return`	`Set` of results of test case executions.

B.1.1.28.12. getRunId()

```
public abstract String getRunId();
```

Get id of test run.

Parameters

`return`	The id of the test run

B.1.1.28.13. getSut()

```
public abstract KlarosSUTImplementation getSut();
```

Get the tested system version.

Parameters

`return`	The SUT version which has been tested

B.1.1.28.14. getTestSuite()

```
public abstract KlarosTestSuite getTestSuite();
```

Get the test suite executed with this test run, if available.

Parameters

`return`	The test suite executed with this test run

B.1.1.28.15. getTimestamp()

```
public abstract Date getTimestamp();
```

Get timestamp.

Parameters

`return`	The time the test run has been executed as `Date` object

B.1.1.28.16. getWork()

```
public abstract Set<KlarosJobTimeBlock> getWork();
```

Get the job time blocks of this test run.

Parameters

`return`	`Set` of job time blocks.

B.1.1.28.17. isPending()

```
public abstract boolean isPending();
```

Checks if the test run is pending. This means that it has not yet been completely executed.

Parameters

`return`	true, if the test run is pending

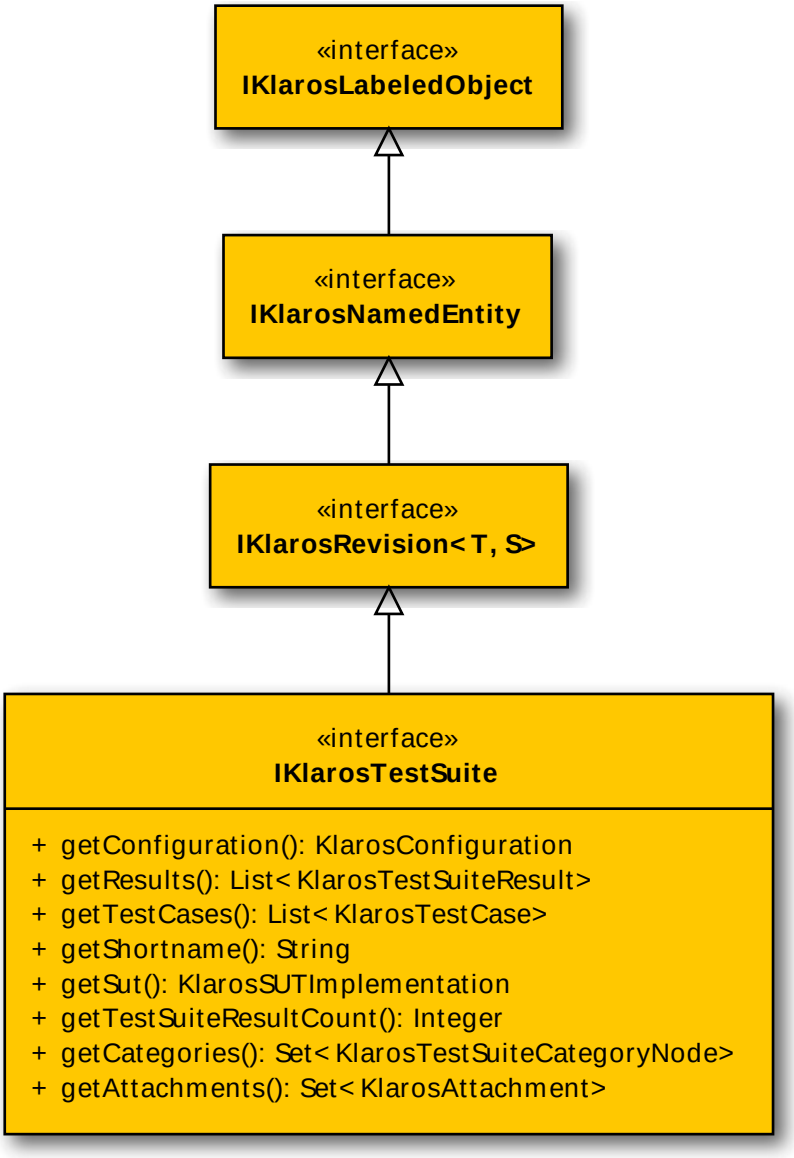
B.1.1.29. IKlarosTestSuite

This interface provides access to data of a test suite.

B.1.1.29.1. Synopsis

```
public interface IKlarosTestSuite<T,S extends Revision> extends IKlarosRevision<T,
 S> {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosTestSuiteCategoryNode> getCategories();
  public abstract KlarosConfiguration getConfiguration();
  public abstract List<KlarosTestSuiteResult> getResults();
  public abstract String getShortname();
  public abstract KlarosSUTImplementation getSut();
  public abstract List<KlarosTestCase> getTestCases();
  public abstract Integer getTestSuiteResultCount();
```

```
}
```



B.1.1.29.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this test suite.

Parameters

`return`	the attachments

B.1.1.29.3. getCategories()

```
public abstract Set<KlarosTestSuiteCategoryNode> getCategories();
```

Gets the categories this object belongs to. Each category node will belong to a different category tree.

Parameters

`return`	the category nodes

B.1.1.29.4. getConfiguration()

```
public abstract KlarosConfiguration getConfiguration();
```

Get configuration.

Parameters

`return`	The related configuration.

B.1.1.29.5. getResults()

```
public abstract List<KlarosTestSuiteResult> getResults();
```

Get test suite results.

Parameters

`return`	`Set` of results of executions of this test suite.

B.1.1.29.6. getShortname()

```
public abstract String getShortname();
```

Get the short name of the test suite.

Parameters

`return`	`String` of the description of the test suite.

B.1.1.29.7. getSut()

```
public abstract KlarosSUTImplementation getSut();
```

Get the description of the test suite.

Parameters

`return`	`String` of the description of the test suite.

B.1.1.29.8. getTestCases()

```
public abstract List<KlarosTestCase> getTestCases();
```

Get the executables of this test suite.

Parameters

`return`	`Set` of test cases of this test suite.

B.1.1.29.9. getTestSuiteResultCount()

```
public abstract Integer getTestSuiteResultCount();
```

Return the number of test suite results in this test suite.

Parameters

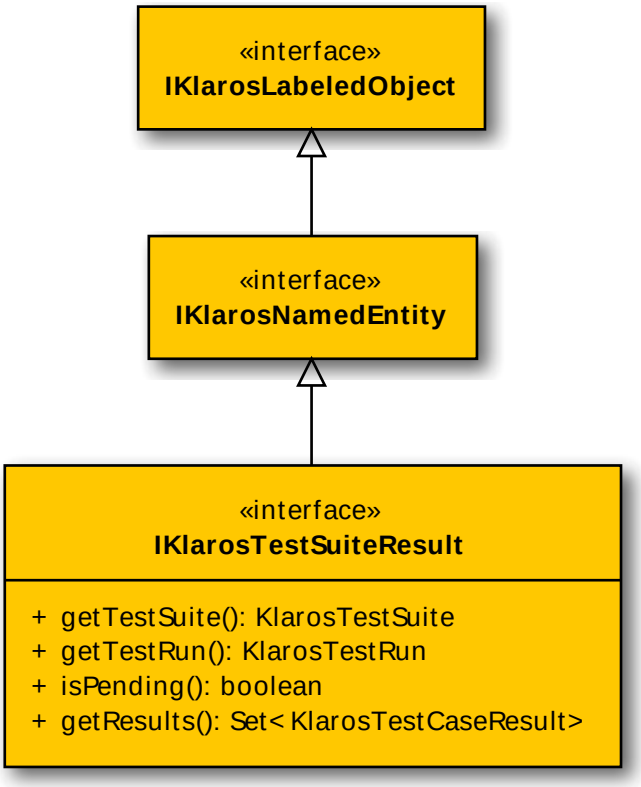
`return`	The number of test suite results in this test suite.

B.1.1.30. IKlarosTestSuiteResult

This interface provides access to data of a test suite result.

B.1.1.30.1. Synopsis

```
public interface IKlarosTestSuiteResult extends IKlarosNamedEntity {

  // Public Methods
  public abstract Set<KlarosTestCaseResult> getResults();
  public abstract KlarosTestRun getTestRun();
  public abstract KlarosTestSuite getTestSuite();
  public abstract boolean isPending();
}
```



B.1.1.30.2. getResults()

```
public abstract Set<KlarosTestCaseResult> getResults();
```

Gets the test case results for this test suite result. The list is sorted by the position in which the test cases appear in the test suite.

Parameters

`return`	the results

B.1.1.30.3. getTestRun()

```
public abstract KlarosTestRun getTestRun();
```

Get the test run for this result.

Parameters

return	the test run

B.1.1.30.4. getTestSuite()

```
public abstract KlarosTestSuite getTestSuite();
```

Get the related test suite.

Parameters

return	The test suite that has been executed to get this result.

B.1.1.30.5. isPending()

```
public abstract boolean isPending();
```

Checks if the test suite result is pending. This means that it has not yet been completely executed.

Parameters

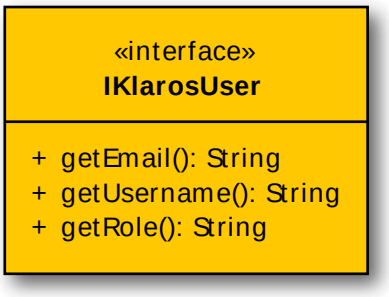
return	true, if the test suite result is pending

B.1.1.31. IKlarosUser

The user object.

B.1.1.31.1. Synopsis

```
public interface IKlarosUser {

  // Public Methods
  public abstract String getEmail();
  public abstract String getRole();
  public abstract String getUsername();
}
```



B.1.1.31.2. getEmail()

```
public abstract String getEmail();
```

The email address of this user.

Parameters

`return`	the email address

B.1.1.31.3. getRole()

```
public abstract String getRole();
```

The role name of this user.

Parameters

`return`	the role name

B.1.1.31.4. getUsername()

```
public abstract String getUsername();
```

The user name of this user as used when logging in.

Parameters

`return`	the user name

B.1.1.32. KlarosAttachment

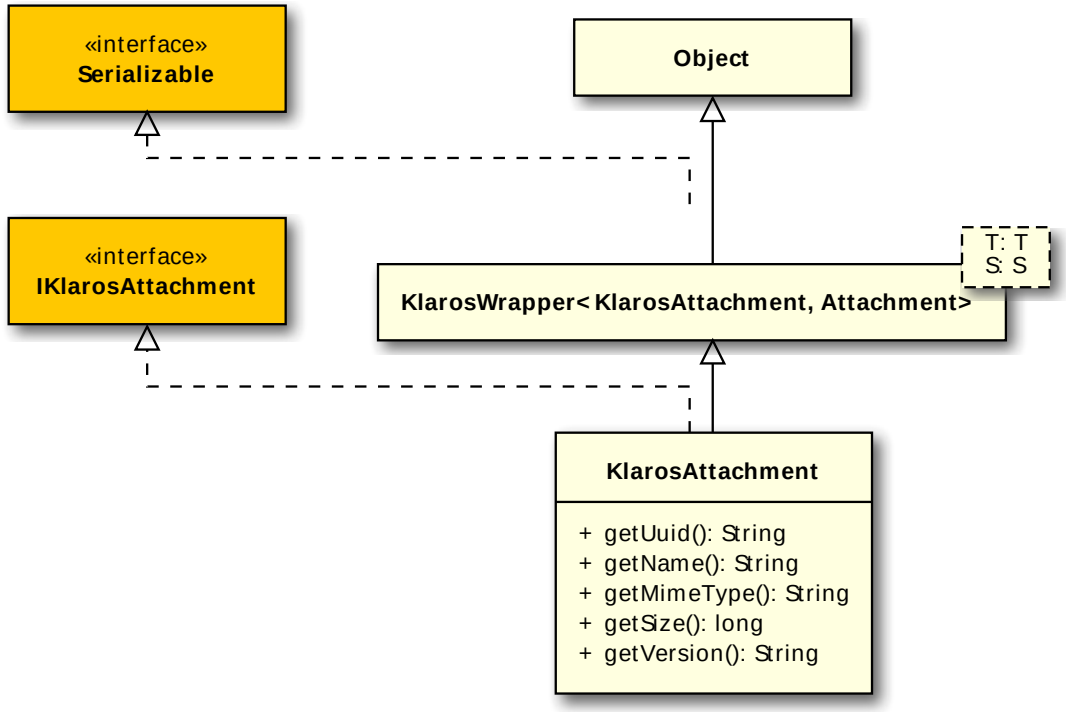
This class provides access to binary attachment.

B.1.1.32.1. Synopsis

```
public final class KlarosAttachment extends KlarosWrapper<KlarosAttachment,
 Attachment>
   implements IKlarosAttachment {

  // Public Methods
  public String getMimeType();
  public String getName();
  public long getSize();
  public String getUuid();
  public String getVersion();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.33. KlarosCategoryNode

A generic category node.

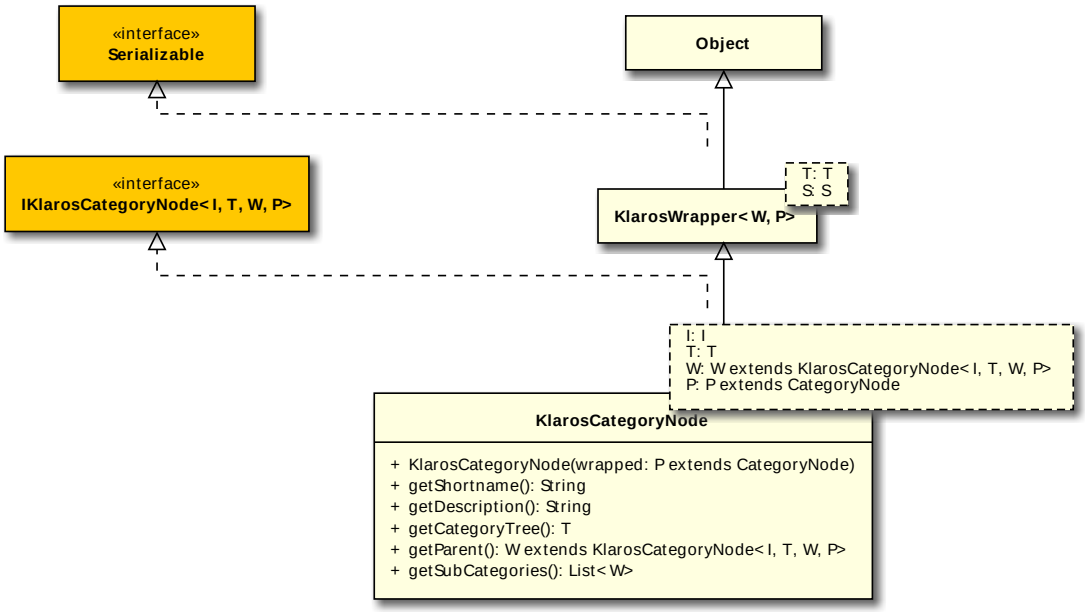
B.1.1.33.1. Synopsis

```
public abstract class KlarosCategoryNode<I,T,W extends KlarosCategoryNode<I, T, W,
 P>,P extends CategoryNode> extends KlarosWrapper<W, P>
   implements IKlarosCategoryNode<I, T, W, P> {

  // Public Constructors
  public KlarosCategoryNode(P wrapped);

  // Public Methods
  public T getCategoryTree();
  public String getDescription();
  public W extends KlarosCategoryNode<I, T, W, P> getParent();
  public String getShortname();
  public List<W> getSubCategories();
}
```

Direct known subclasses: de.verit.klaros.core.model.KlarosIterationCategoryNode , de.verit.klaros.core.model.KlarosRequirementCategoryNode , de.verit.klaros.core.model.KlarosSUTImplementationCategoryNode , de.verit.klaros.core.model.KlarosTestCaseCategoryNode , de.verit.klaros.core.model.KlarosTestEnvironmentCategoryNode , de.verit.klaros.core.model.KlarosTestSuiteCategoryNode

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: getWrapped

Methods inherited from java.lang.Object: equals , getClass , hashCode , notify , notifyAll , toString , wait



B.1.1.33.2. KlarosCategoryNode(P)

```
public KlarosCategoryNode(P wrapped);
```

Instantiates a new klaros category node.

Parameters

`wrapped`	the wrapped class

B.1.1.34. KlarosCategoryTree

A generic category tree.

B.1.1.34.1. Synopsis

```
public class KlarosCategoryTree<I,W extends KlarosCategoryTree<I, W, P, X,
 Y>,P extends CategoryTree,X extends KlarosCategoryNode<I, W, X, Y>,Y extends
 CategoryNode> extends
                                KlarosNamedEntity<W, P>

  implements IKlarosCategoryTree<I, W, P, X, Y> {

 // Public Methods
 public String getDescription();
 public KlarosCategoryNode<I, W, X, Y> getRootNode();
 public String getShortname();
}
```

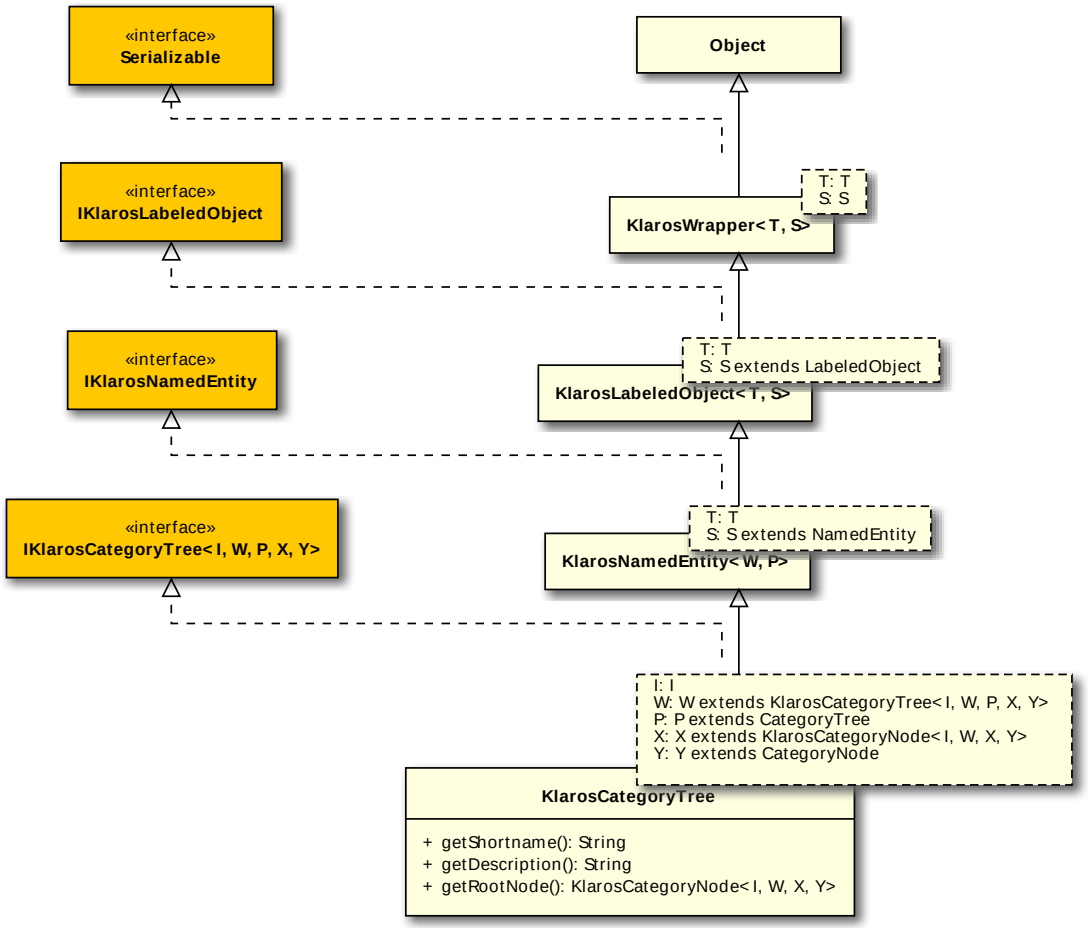
Direct known subclasses: de.verit.klaros.core.model.KlarosIterationCategoryTree , de.verit.klaros.core.model.KlarosRequirementCategoryTree , de.verit.klaros.core.model.KlarosSUTImplementationCategoryTree , de.verit.klaros.core.model.KlarosTestCaseCategoryTree , de.verit.klaros.core.model.KlarosTestEnvironmentCategoryTree , de.verit.klaros.core.model.KlarosTestSuiteCategoryTree

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.35. KlarosConfiguration

This class provides access to the information stored for project's configuration.

B.1.1.35.1. Synopsis

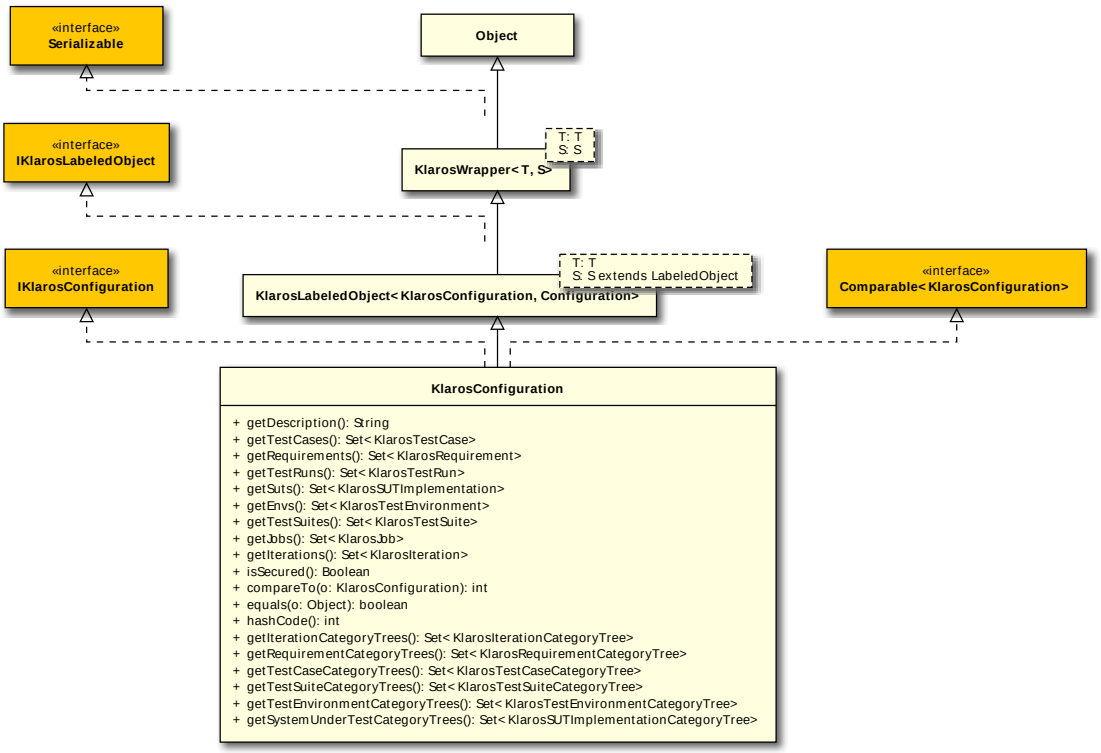
```
public class KlarosConfiguration extends
                              KlarosLabeledObject<KlarosConfiguration,
 Configuration>

  implements IKlarosConfiguration,
         Comparable<KlarosConfiguration> {

 // Public Methods
```

```
  public int compareTo(KlarosConfiguration o);
  public boolean equals(Object o);
  public String getDescription();
  public Set<KlarosTestEnvironment> getEnvs();
  public Set<KlarosIterationCategoryTree> getIterationCategoryTrees();
  public Set<KlarosIteration> getIterations();
  public Set<KlarosJob> getJobs();
  public Set<KlarosRequirementCategoryTree> getRequirementCategoryTrees();
  public Set<KlarosRequirement> getRequirements();
  public Set<KlarosSUTImplementation> getSuts();
  public Set<KlarosSUTImplementationCategoryTree> getSystemUnderTestCategoryTrees();
  public Set<KlarosTestCaseCategoryTree> getTestCaseCategoryTrees();
  public Set<KlarosTestCase> getTestCases();
  public Set<KlarosTestEnvironmentCategoryTree> getTestEnvironmentCategoryTrees();
  public Set<KlarosTestRun> getTestRuns();
  public Set<KlarosTestSuiteCategoryTree> getTestSuiteCategoryTrees();
  public Set<KlarosTestSuite> getTestSuites();
  public int hashCode();
  public Boolean isSecured();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.36. KlarosEnumValue

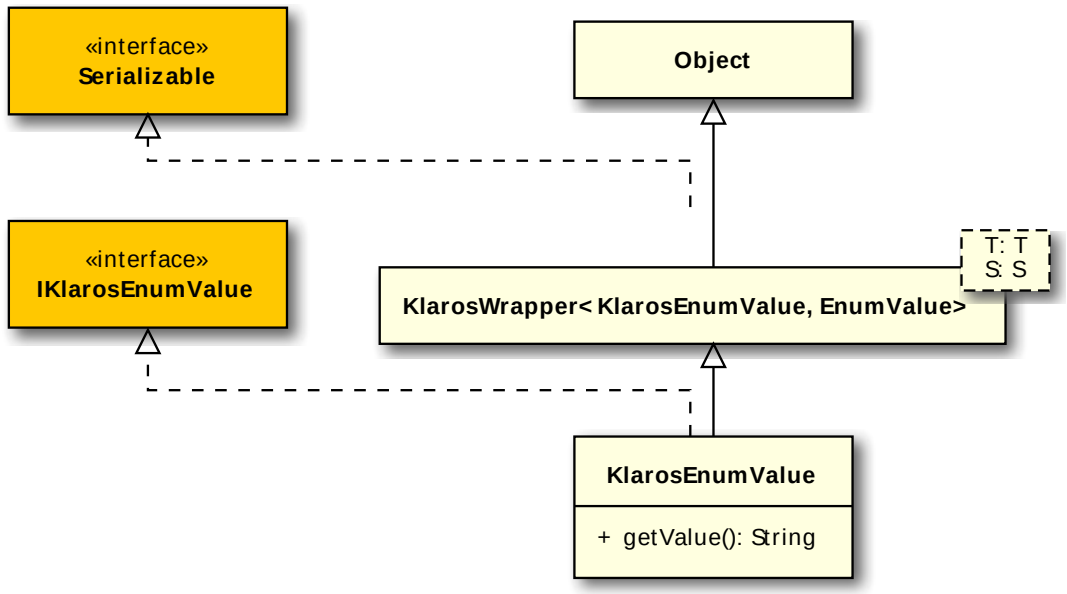
A Custom property enumeration value.

B.1.1.36.1. Synopsis

```
public final class KlarosEnumValue extends KlarosWrapper<KlarosEnumValue, EnumValue>
   implements IKlarosEnumValue {

  // Public Methods
  public String getValue();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.37. KlarosExternalImplementation

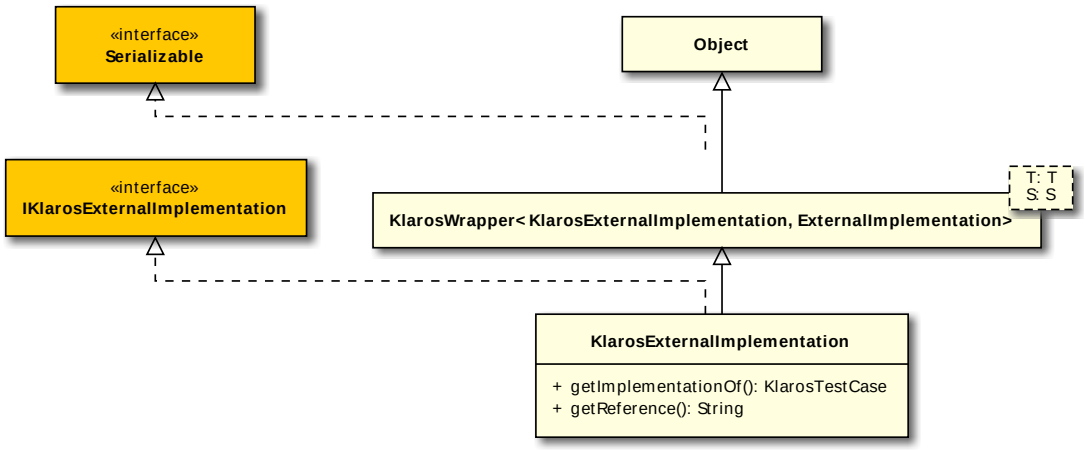
This class encapsulates external implementation data of a test case.

B.1.1.37.1. Synopsis

```
public final class KlarosExternalImplementation
 extends KlarosWrapper<KlarosExternalImplementation, ExternalImplementation>
   implements IKlarosExternalImplementation {

  // Public Methods
  public KlarosTestCase getImplementationOf();
  public String getReference();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.38. KlarosIssue

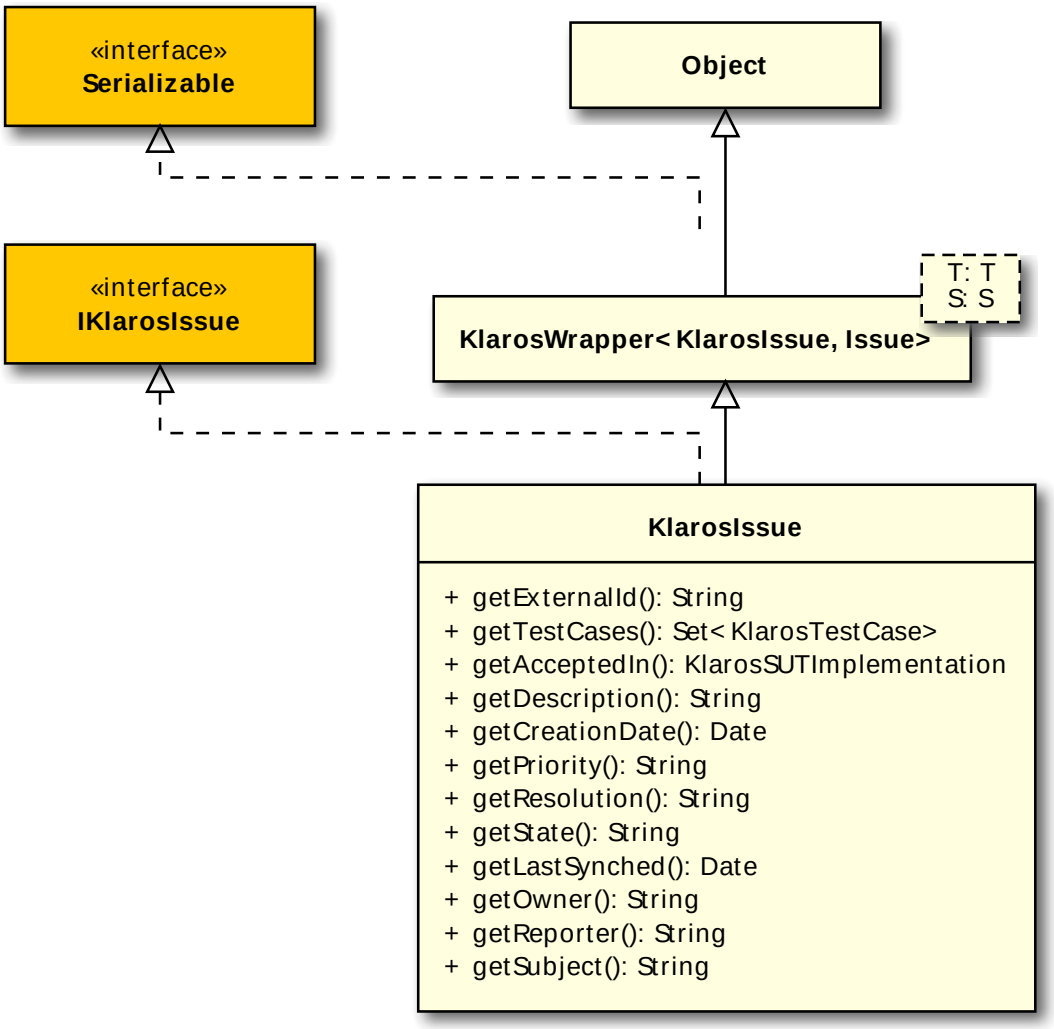
This class provides access to the information stored for detected issues.

B.1.1.38.1. Synopsis

```
public final class KlarosIssue extends KlarosWrapper<KlarosIssue, Issue>
   implements IKlarosIssue {

  // Public Methods
  public KlarosSUTImplementation getAcceptedIn();
  public Date getCreationDate();
  public String getDescription();
  public String getExternalId();
  public Date getLastSynched();
  public String getOwner();
  public String getPriority();
  public String getReporter();
  public String getResolution();
  public String getState();
  public String getSubject();
  public Set<KlarosTestCase> getTestCases();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.39. KlarosIteration

An iteration in a project.

B.1.1.39.1. Synopsis

```
public final class KlarosIteration extends
                                KlarosNamedEntity<KlarosIteration, Iteration>

  implements IKlarosIteration,
         Comparable<KlarosIteration> {

 // Public Methods
 public int compareTo(KlarosIteration o);
 public boolean equals(Object o);
 public Set<KlarosAttachment> getAttachments();
 public Set<KlarosIterationCategoryNode> getCategories();
 public StateDef getCurrentState();
 public String getDescription();
 public Date getDueDate();
 public Set<KlarosTestEnvironment> getEnvs();
```

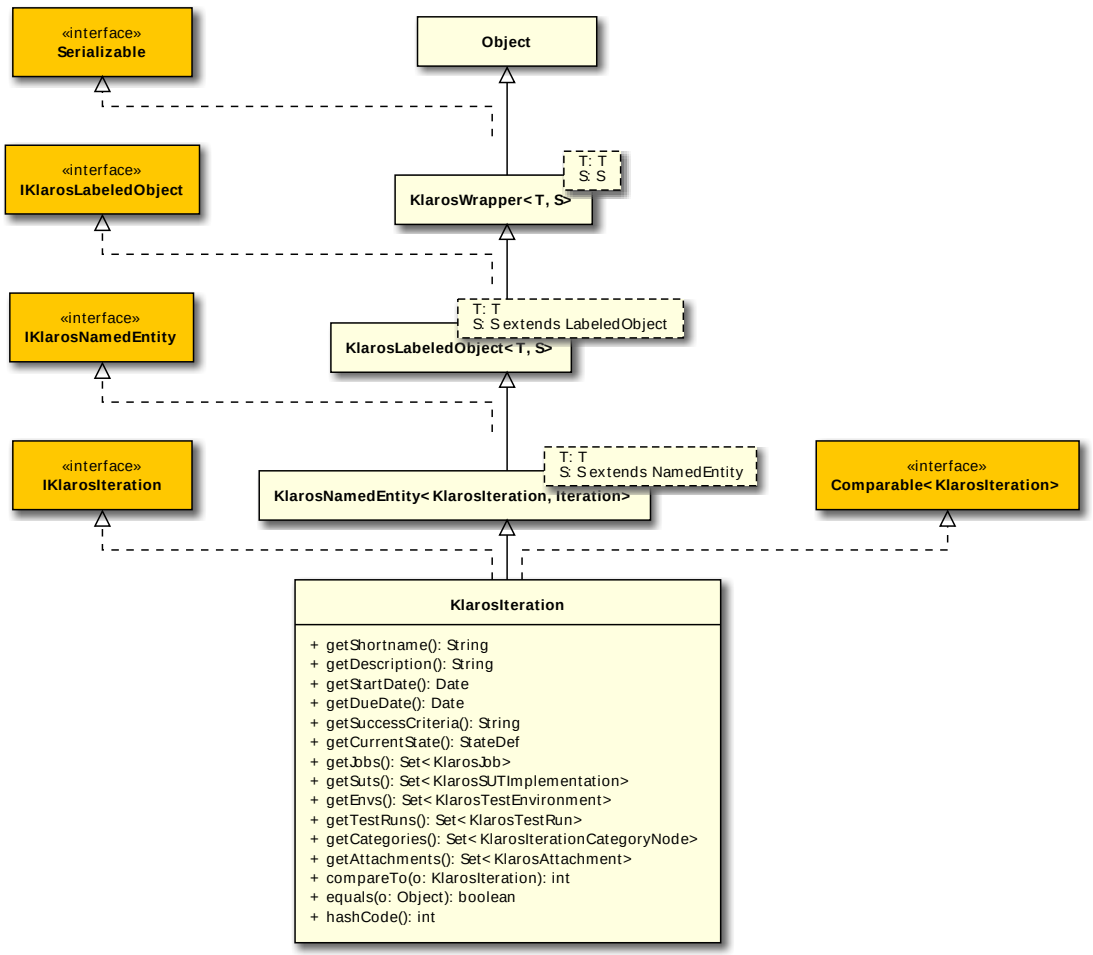
```
  public Set<KlarosJob> getJobs();
  public String getShortname();
  public Date getStartDate();
  public String getSuccessCriteria();
  public Set<KlarosSUTImplementation> getSuts();
  public Set<KlarosTestRun> getTestRuns();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.40. KlarosIterationCategoryNode

An category node for iteration implementations.

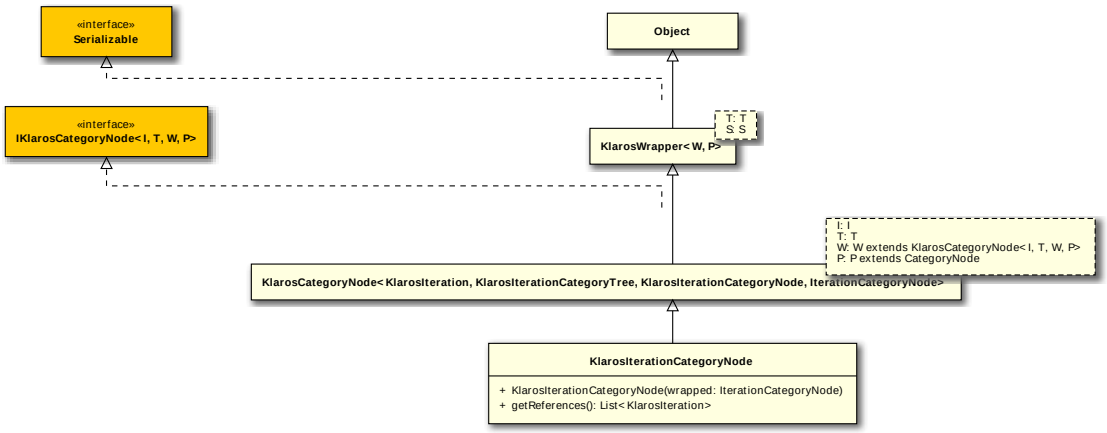
B.1.1.40.1. Synopsis

```
public final class KlarosIterationCategoryNode extends
                                KlarosCategoryNode<KlarosIteration,
 KlarosIterationCategoryTree, KlarosIterationCategoryNode, IterationCategoryNode>
                               {

  // Public Constructors
  public KlarosIterationCategoryNode(IterationCategoryNode wrapped);

  // Public Methods
  public List<KlarosIteration> getReferences();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosCategoryNode: `getCategoryTree` , `getDescription` , `getParent` , `getShortname` , `getSubCategories`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.40.2. KlarosIterationCategoryNode(IterationCategoryNode)

```
public KlarosIterationCategoryNode(IterationCategoryNode wrapped);
```

Instantiates a new klaros iteration category node.

Parameters

`wrapped`	the wrapped class

B.1.1.41. KlarosIterationCategoryTree

An category tree for Iteration implementations.

B.1.1.41.1. Synopsis

```
public class KlarosIterationCategoryTree extends
```

```
                                KlarosCategoryTree<KlarosIteration,
 KlarosIterationCategoryTree, IterationCategoryTree, KlarosIterationCategoryNode,
 IterationCategoryNode>
                              {
}
```

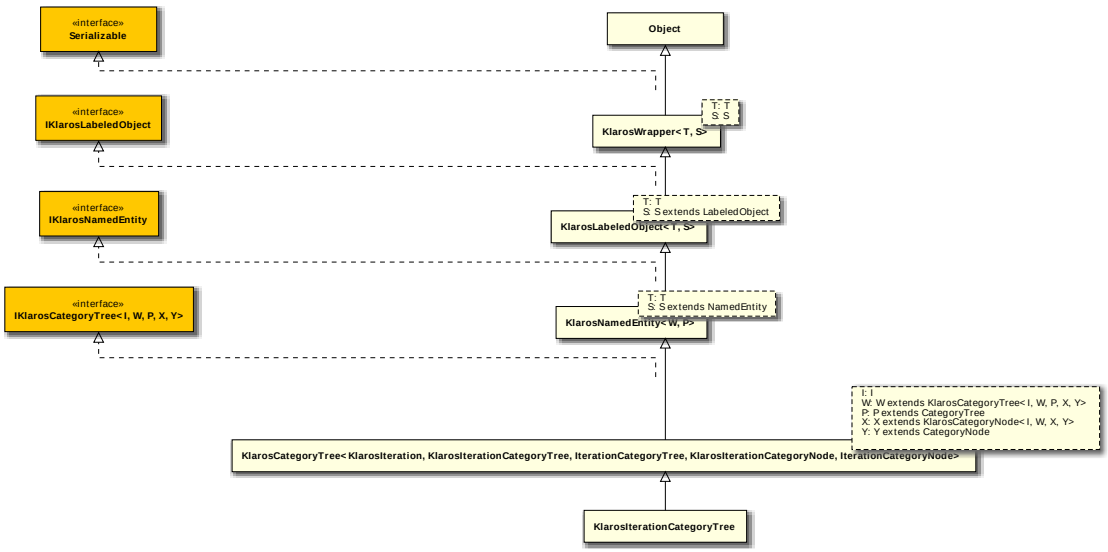
Methods inherited from de.verit.klaros.core.model.KlarosCategoryTree: `getDescription` , `getRootNode` , `getShortname`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.42. KlarosJob

This class provides access to the information stored for jobs.

B.1.1.42.1. Synopsis

```
public final class KlarosJob extends
                                KlarosNamedEntity<KlarosJob, Job>

  implements IKlarosJob,
         Comparable<KlarosJob> {

 // Public Methods
 public int compareTo(KlarosJob o);
 public boolean equals(Object o);
```

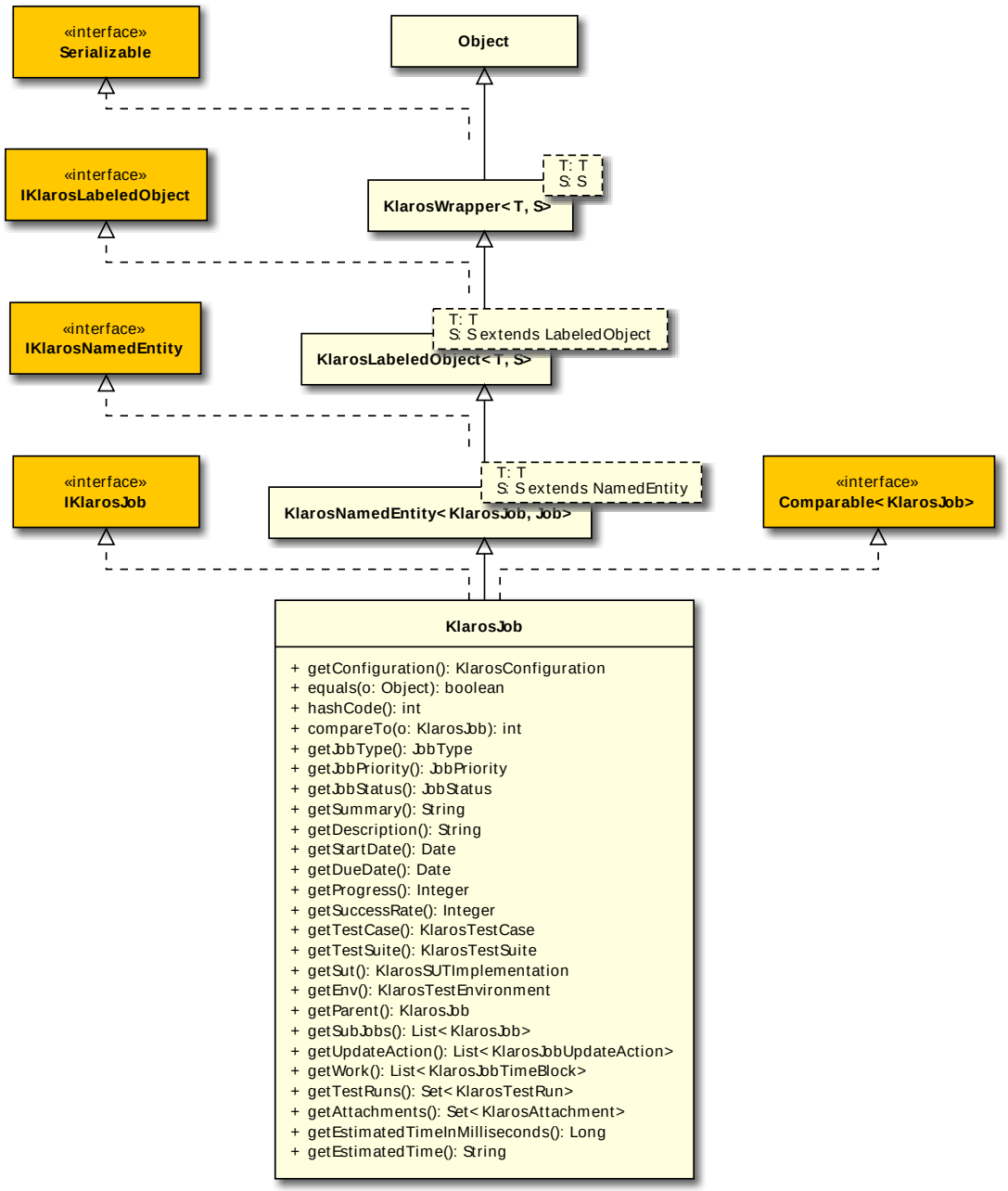
```
  public Set<KlarosAttachment> getAttachments();
  public KlarosConfiguration getConfiguration();
  public String getDescription();
  public Date getDueDate();
  public KlarosTestEnvironment getEnv();
  public String getEstimatedTime();
  public Long getEstimatedTimeInMilliseconds();
  public JobPriority getJobPriority();
  public JobStatus getJobStatus();
  public JobType getJobType();
  public KlarosJob getParent();
  public Integer getProgress();
  public Date getStartDate();
  public List<KlarosJob> getSubJobs();
  public Integer getSuccessRate();
  public String getSummary();
  public KlarosSUTImplementation getSut();
  public KlarosTestCase getTestCase();
  public Set<KlarosTestRun> getTestRuns();
  public KlarosTestSuite getTestSuite();
  public List<KlarosJobUpdateAction> getUpdateAction();
  public List<KlarosJobTimeBlock> getWork();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.43. KlarosJobTimeBlock

The implementation of a job time block, measuring a unit of work done for a job.

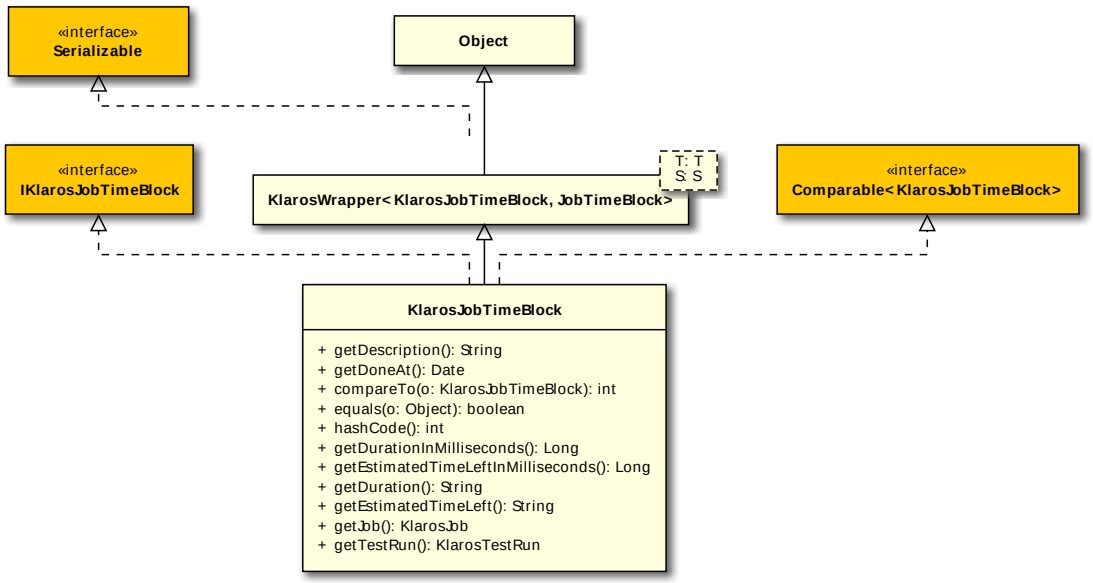
B.1.1.43.1. Synopsis

```
public final class KlarosJobTimeBlock extends KlarosWrapper<KlarosJobTimeBlock,
 JobTimeBlock>
  implements IKlarosJobTimeBlock,
          Comparable<KlarosJobTimeBlock> {

 // Public Methods
 public int compareTo(KlarosJobTimeBlock o);
 public boolean equals(Object o);
```

```
  public String getDescription();
  public Date getDoneAt();
  public String getDuration();
  public Long getDurationInMilliseconds();
  public String getEstimatedTimeLeft();
  public Long getEstimatedTimeLeftInMilliseconds();
  public KlarosJob getJob();
  public KlarosTestRun getTestRun();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.44. KlarosJobUpdateAction

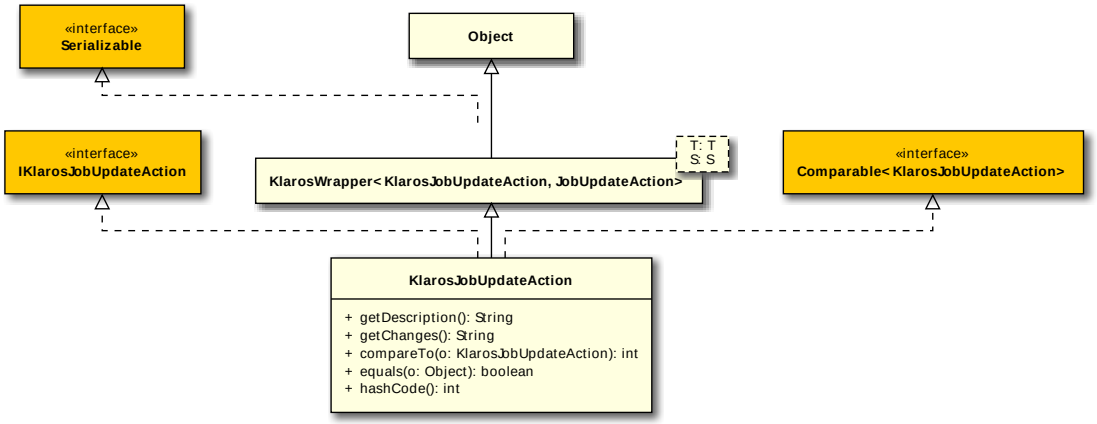
The model of a job comment.

B.1.1.44.1. Synopsis

```
public final class KlarosJobUpdateAction extends KlarosWrapper<KlarosJobUpdateAction,
 JobUpdateAction>
   implements IKlarosJobUpdateAction,
           Comparable<KlarosJobUpdateAction> {

  // Public Methods
  public int compareTo(KlarosJobUpdateAction o);
  public boolean equals(Object o);
  public String getChanges();
  public String getDescription();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.45. KlarosLabeledObject

This class encapsulates the (dynamic) properties of a klaros object.

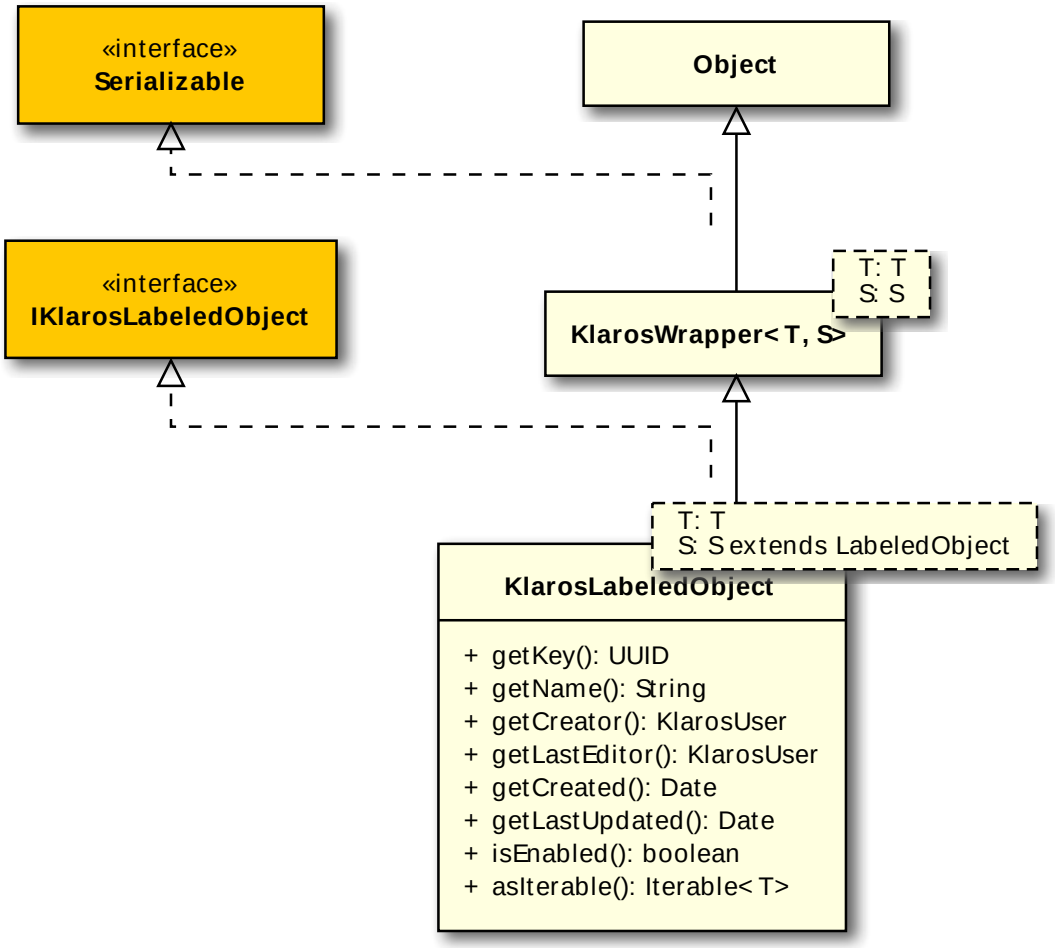
B.1.1.45.1. Synopsis

```
public abstract class KlarosLabeledObject<T,S extends LabeledObject>
 extends KlarosWrapper<T, S>
   implements IKlarosLabeledObject {

  // Public Methods
  public Iterable<T> asIterable();
  public Date getCreated();
  public KlarosUser getCreator();
  public UUID getKey();
  public KlarosUser getLastEditor();
  public Date getLastUpdated();
  public String getName();
  public boolean isEnabled();
}
```

Direct known subclasses: `de.verit.klaros.core.model.KlarosConfiguration` , `de.verit.klaros.core.model.KlarosNamedEntity` , `de.verit.klaros.core.model.KlarosUser`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.46. KlarosNamedEntity

This class encapsulates the (dynamic) properties of a klaros object.

B.1.1.46.1. Synopsis

```
public abstract class KlarosNamedEntity<T,S extends NamedEntity> extends
                                KlarosLabeledObject<T, S>

   implements IKlarosNamedEntity {

  // Public Methods
  public KlarosUser getAssignee();
  public List<IKlarosProperty> getProperties();
  public boolean isDefinedProperty(String propertyName);
}
```

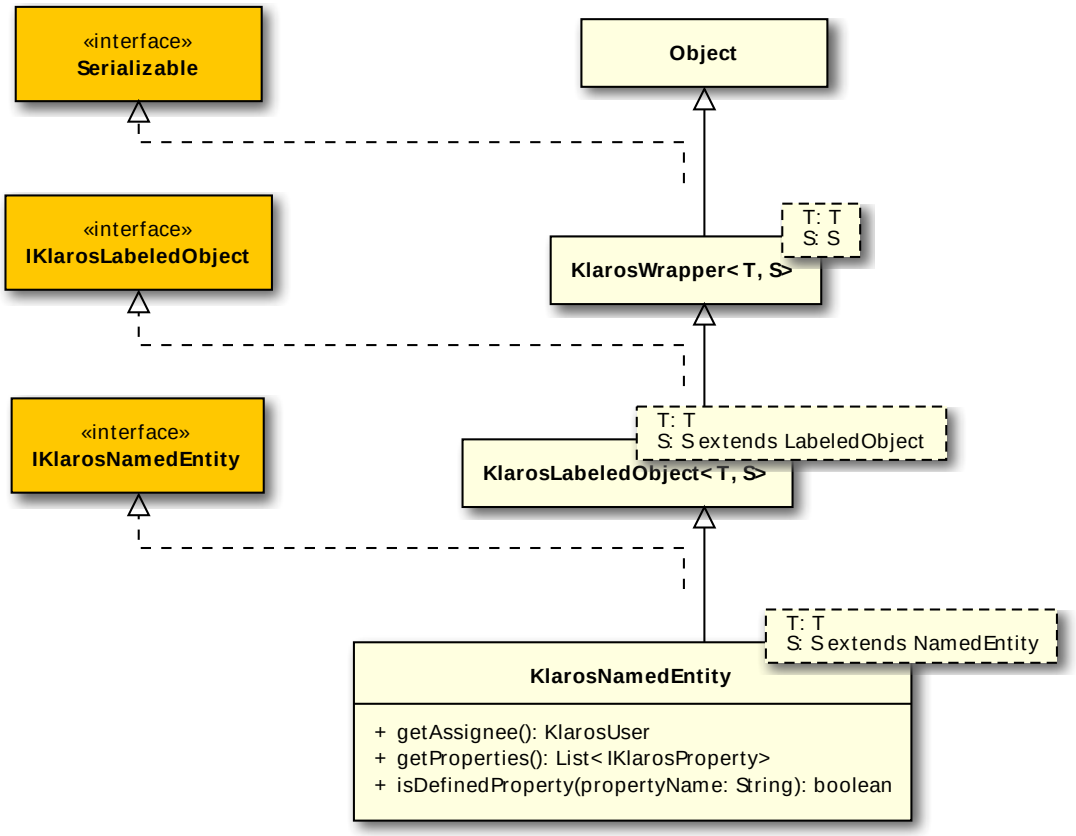
Direct known subclasses: de.verit.klaros.core.model.KlarosCategoryTree , de.verit.klaros.core.model.KlarosIteration , de.verit.klaros.core.model.KlarosJob , de.verit.klaros.core.model.KlarosRequirementGroup , de.verit.klaros.core.model.KlarosRevision , de.verit.klaros.core.model.KlarosSUTImplementation , de.verit.klaros.core.model.KlarosTestCaseResult , de.verit.klaros.core.model.KlarosTestCaseStepResult , de.-

verit.klaros.core.model.KlarosTestEnvironment , de.verit.klaros.core.model.KlarosTestRun , de.verit.klaros.core.model.KlarosTestSuiteResult

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: getWrapped

Methods inherited from java.lang.Object: equals , getClass , hashCode , notify , notifyAll , toString , wait



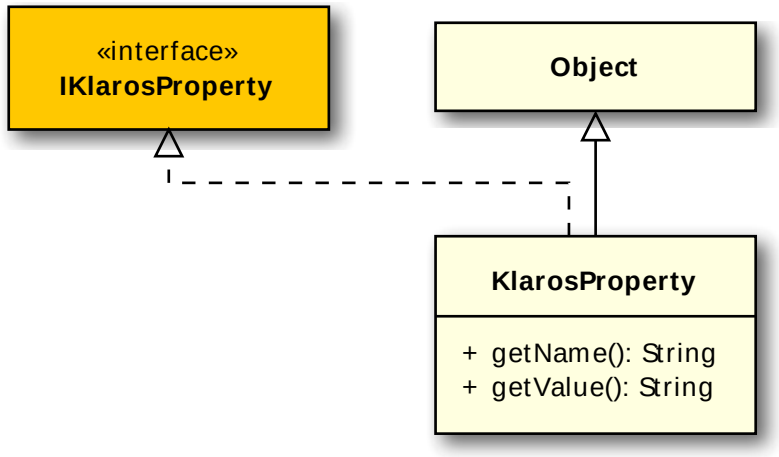
B.1.1.47. KlarosProperty

The Klaros property class.

B.1.1.47.1. Synopsis

```
public final class KlarosProperty implements IKlarosProperty {

  // Public Methods
  public String getName();
  public String getValue();
}
```

Methods inherited from java.lang.Object: equals , getClass , hashCode , notify , notifyAll , toString , wait



B.1.1.48. KlarosRequirement

This class provides access to the information stored for requirements.

B.1.1.48.1. Synopsis

```
public final class KlarosRequirement extends
                                KlarosRevision<KlarosRequirement, Requirement>

   implements IKlarosRequirement<KlarosRequirement, Requirement> {

  // Public Methods
  public Set<KlarosAttachment> getAttachments();
  public KlarosRequirement getBranchRoot();
  public Set<KlarosRevision<KlarosRequirement, Requirement>> getBranches();
  public Set<KlarosRequirementCategoryNode> getCategories();
  public KlarosConfiguration getConfiguration();
  public Set<KlarosTestCase> getCoveringTestCases();
  public String getDescription();
  public Set<String> getExternalNames();
  public Set<KlarosIteration> getIterations();
  public KlarosRequirement getPredecessor();
  public RequirementPriority getPriority();
  public KlarosRequirementGroup getRequirementGroup();
  public KlarosRequirement getRoot();
  public String getShortname();
  public String getState();
  public KlarosRequirement getSuccessor();
  public String getSummary();
  public KlarosRequirement getTrunkRoot();
}
```

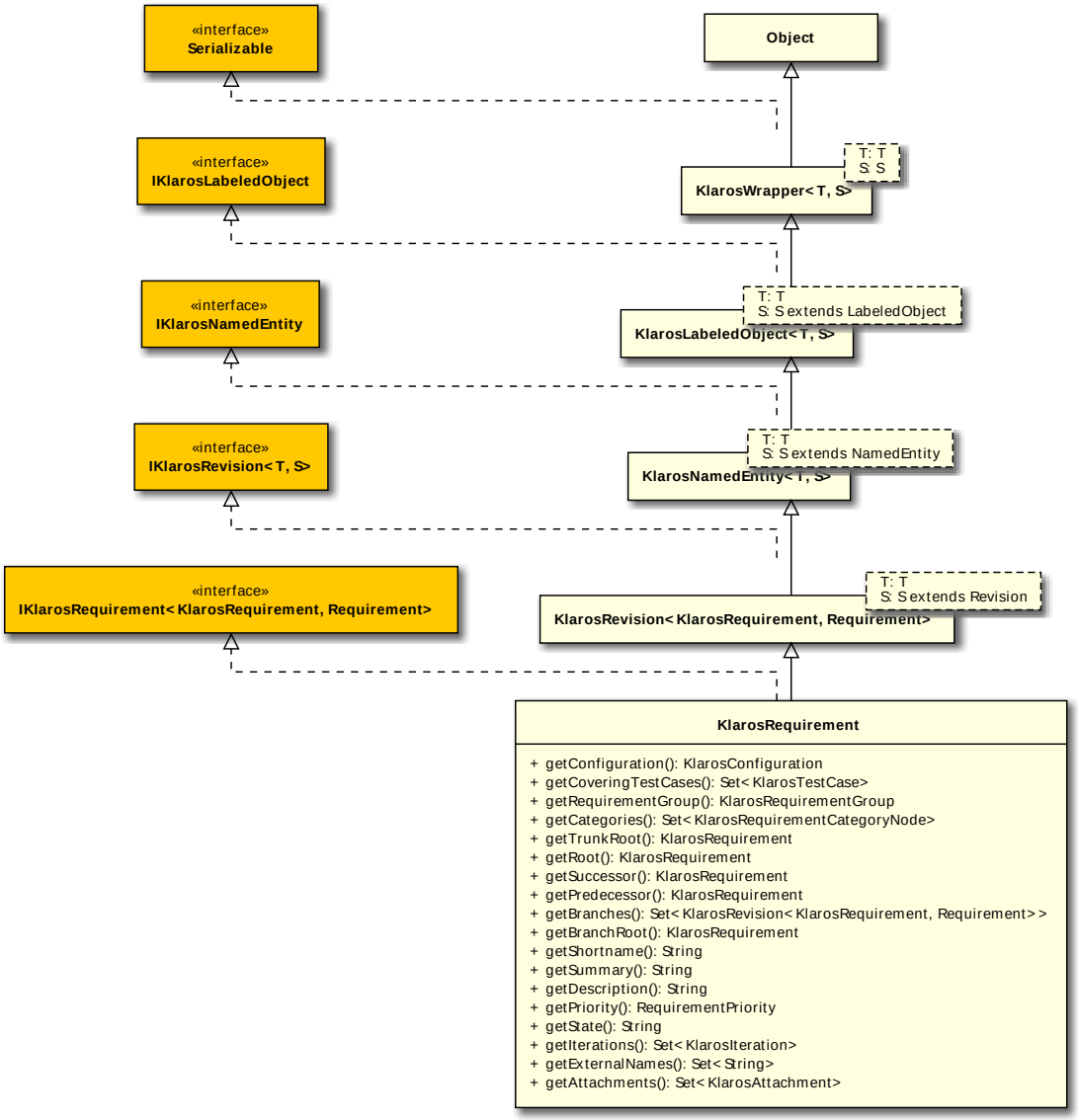
Methods inherited from de.verit.klaros.core.model.KlarosRevision: `getRevisionComment` , `getRevisionId`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.49. KlarosRequirementCategoryNode

An category node for requirement implementations.

B.1.1.49.1. Synopsis

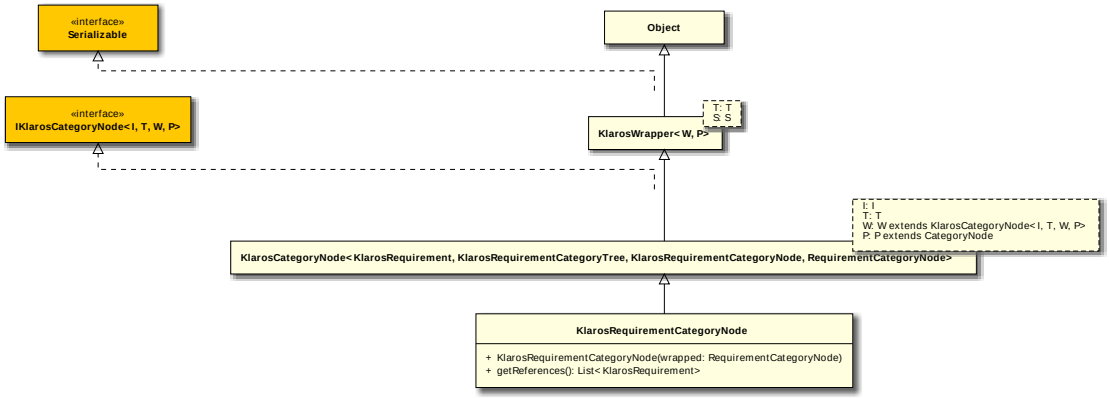
```
public final class KlarosRequirementCategoryNode extends
                            KlarosCategoryNode<KlarosRequirement,
 KlarosRequirementCategoryTree, KlarosRequirementCategoryNode,
 RequirementCategoryNode>
```

```
                                {

  // Public Constructors
  public KlarosRequirementCategoryNode(RequirementCategoryNode wrapped);

  // Public Methods
  public List<KlarosRequirement> getReferences();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosCategoryNode: `getCategoryTree` , `getDescription` , `getParent` , `getShortname` , `getSubCategories`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.49.2. KlarosRequirementCategoryNode(RequirementCategoryNode)

```
public KlarosRequirementCategoryNode(RequirementCategoryNode wrapped);
```

Instantiates a new klaros requirement category node.

Parameters

`wrapped`	the wrapped class

B.1.1.50. KlarosRequirementCategoryTree

An category tree for Requirement implementations.

B.1.1.50.1. Synopsis

```
public final class KlarosRequirementCategoryTree extends
                                KlarosCategoryTree<KlarosRequirement,
 KlarosRequirementCategoryTree, RequirementCategoryTree,
 KlarosRequirementCategoryNode, RequirementCategoryNode>
                                {
}
```

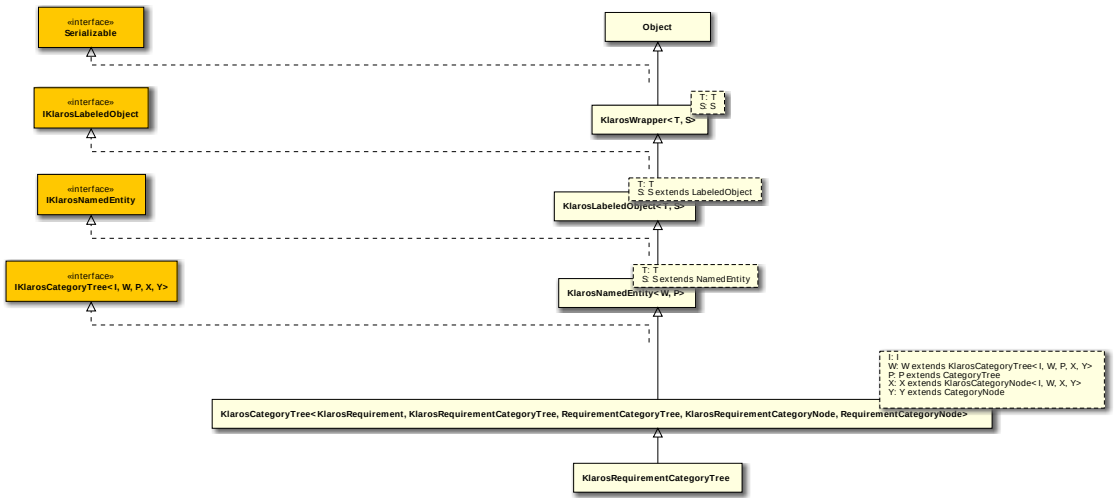
Methods inherited from de.verit.klaros.core.model.KlarosCategoryTree: `getDescription` , `getRootNode` , `getShortname`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.51. KlarosRequirementGroup

This class realizes the grouping of test requirements.

B.1.1.51.1. Synopsis

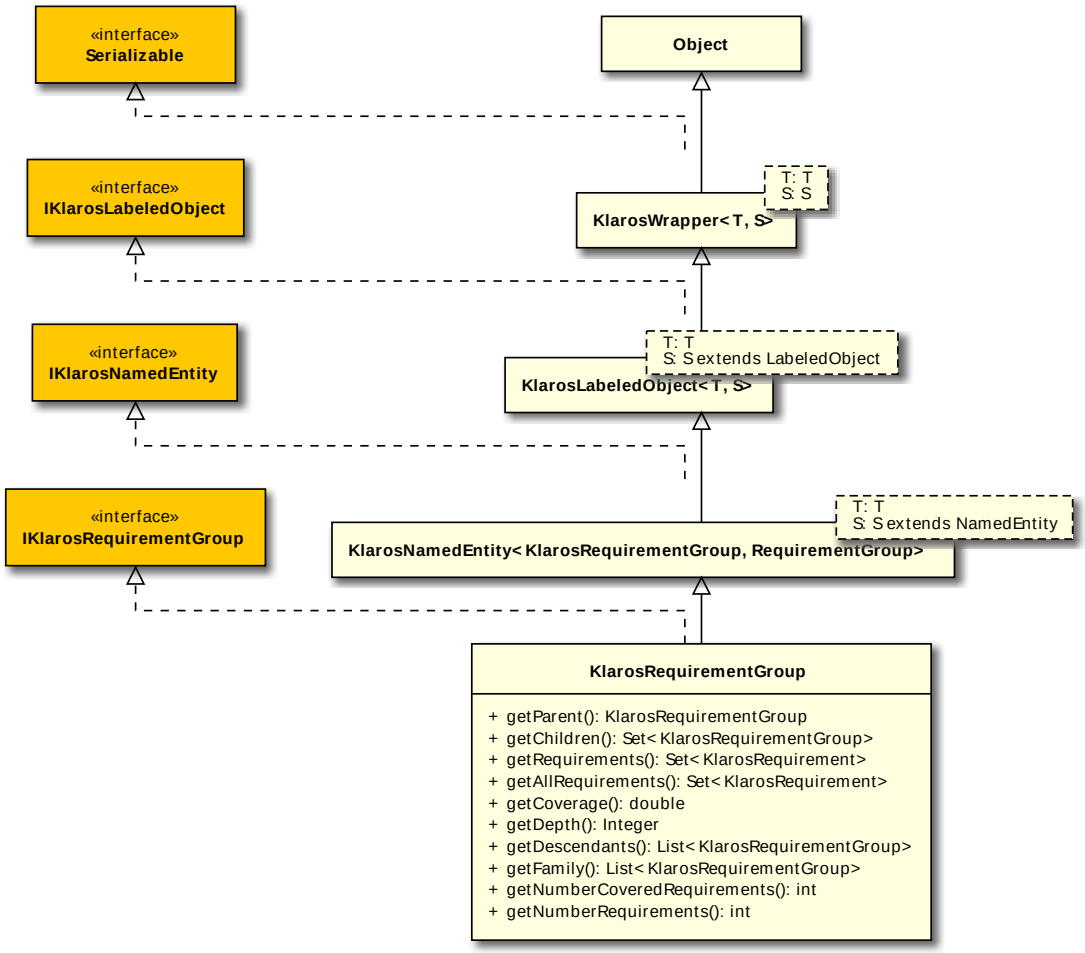
```
public final class KlarosRequirementGroup extends
                                KlarosNamedEntity<KlarosRequirementGroup,
 RequirementGroup>

  implements IKlarosRequirementGroup {

 // Public Methods
 public Set<KlarosRequirement> getAllRequirements();
 public Set<KlarosRequirementGroup> getChildren();
 public double getCoverage();
 public Integer getDepth();
 public List<KlarosRequirementGroup> getDescendants();
 public List<KlarosRequirementGroup> getFamily();
 public int getNumberCoveredRequirements();
 public int getNumberRequirements();
 public KlarosRequirementGroup getParent();
 public Set<KlarosRequirement> getRequirements();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: getWrapped

Methods inherited from java.lang.Object: equals , getClass , hashCode , notify , notifyAll , toString , wait



B.1.1.52. KlarosRevision

This class encapsulates the revision related information of a klaros object.

B.1.1.52.1. Synopsis

```
public abstract class KlarosRevision<T,S extends Revision> extends
                                KlarosNamedEntity<T, S>

   implements IKlarosRevision<T, S> {

  // Public Methods
  public final String getRevisionComment();
  public final String getRevisionId();
}
```

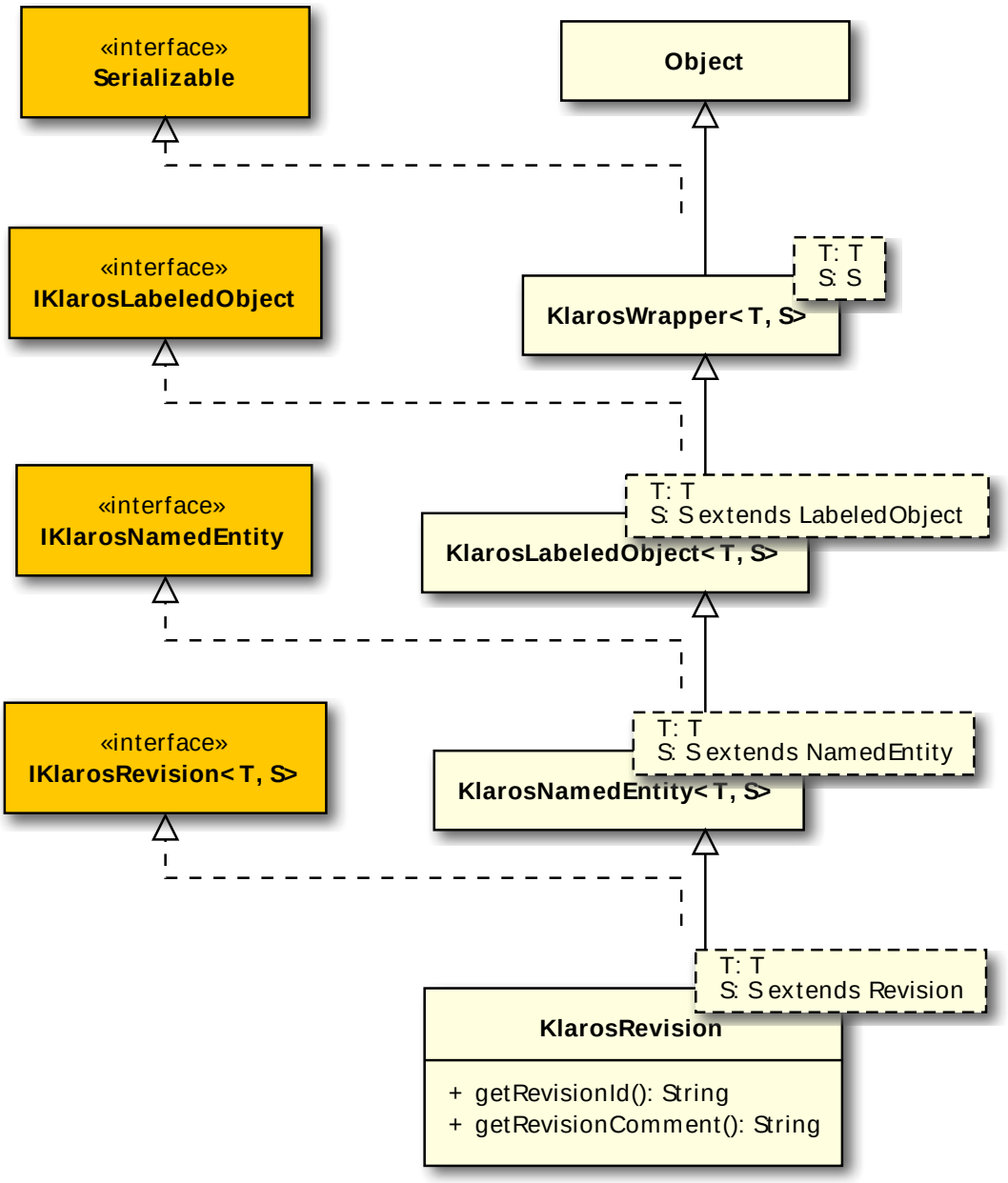
Direct known subclasses: `de.verit.klaros.core.model.KlarosRequirement` , `de.verit.klaros.core.model.KlarosTestCase` , `de.verit.klaros.core.model.KlarosTestSuite`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.53. KlarosSUTImplementation

This class provides access to the information stored for systems under tests (SUT).

B.1.1.53.1. Synopsis

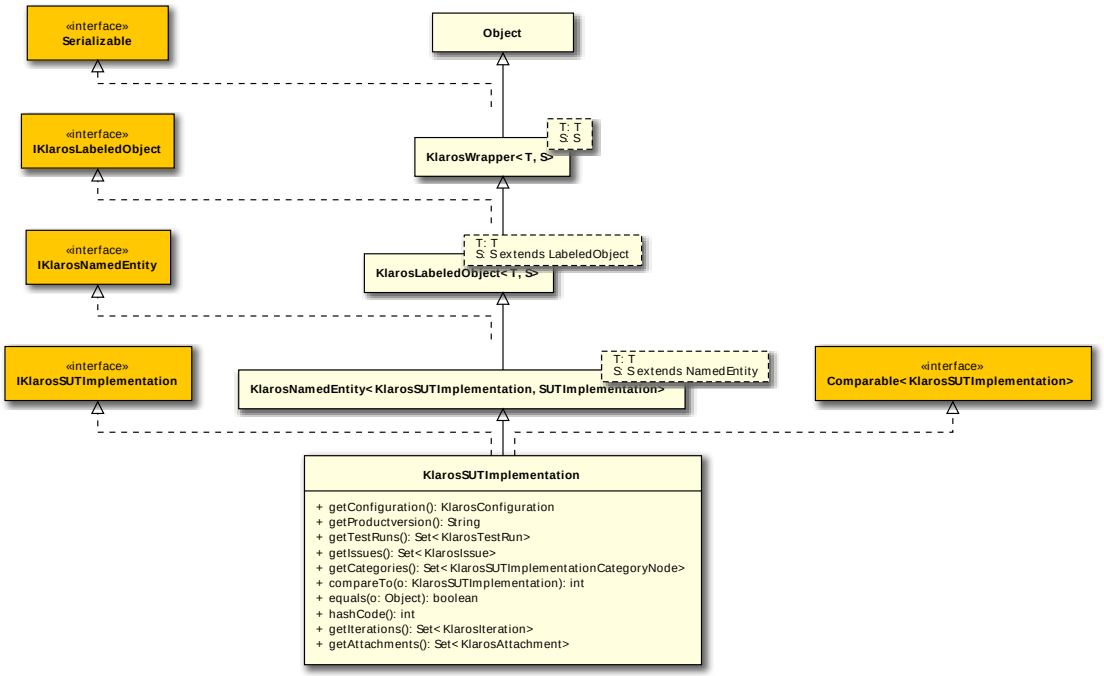
```
public final class KlarosSUTImplementation extends
                                KlarosNamedEntity<KlarosSUTImplementation,
 SUTImplementation>

  implements IKlarosSUTImplementation,
         Comparable<KlarosSUTImplementation> {

  // Public Methods
  public int compareTo(KlarosSUTImplementation o);
  public boolean equals(Object o);
  public Set<KlarosAttachment> getAttachments();
  public Set<KlarosSUTImplementationCategoryNode> getCategories();
  public KlarosConfiguration getConfiguration();
  public Set<KlarosIssue> getIssues();
  public Set<KlarosIteration> getIterations();
  public String getProductversion();
  public Set<KlarosTestRun> getTestRuns();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.54. KlarosSUTImplementationCategoryNode

An category node for SUT implementations.

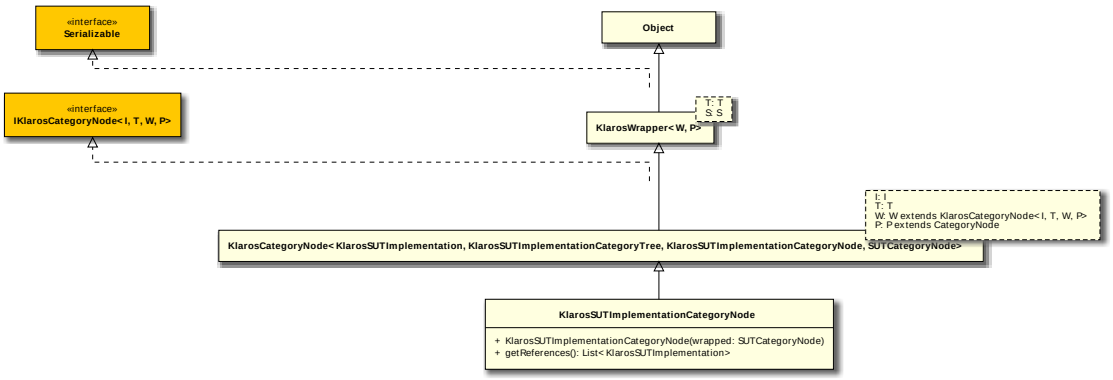
B.1.1.54.1. Synopsis

```
public final class KlarosSUTImplementationCategoryNode extends
                                KlarosCategoryNode<KlarosSUTImplementation,
 KlarosSUTImplementationCategoryTree, KlarosSUTImplementationCategoryNode,
 SUTCategoryNode>
                               {

  // Public Constructors
  public KlarosSUTImplementationCategoryNode(SUTCategoryNode wrapped);

  // Public Methods
  public List<KlarosSUTImplementation> getReferences();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosCategoryNode: `getCategoryTree` , `getDescription` , `getParent` , `getShortname` , `getSubCategories`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.54.2. KlarosSUTImplementationCategoryNode(SUTCategoryNode)

```
public KlarosSUTImplementationCategoryNode(SUTCategoryNode wrapped);
```

Instantiates a new klaros sut implementation category node.

Parameters

`wrapped`	the wrapped class

B.1.1.55. KlarosSUTImplementationCategoryTree

An category tree for SUT implementations.

B.1.1.55.1. Synopsis

```
public final class KlarosSUTImplementationCategoryTree extends
```

```
                                KlarosCategoryTree<KlarosSUTImplementation,
 KlarosSUTImplementationCategoryTree, SUTCategoryTree,
 KlarosSUTImplementationCategoryNode, SUTCategoryNode>
                              {
}
```

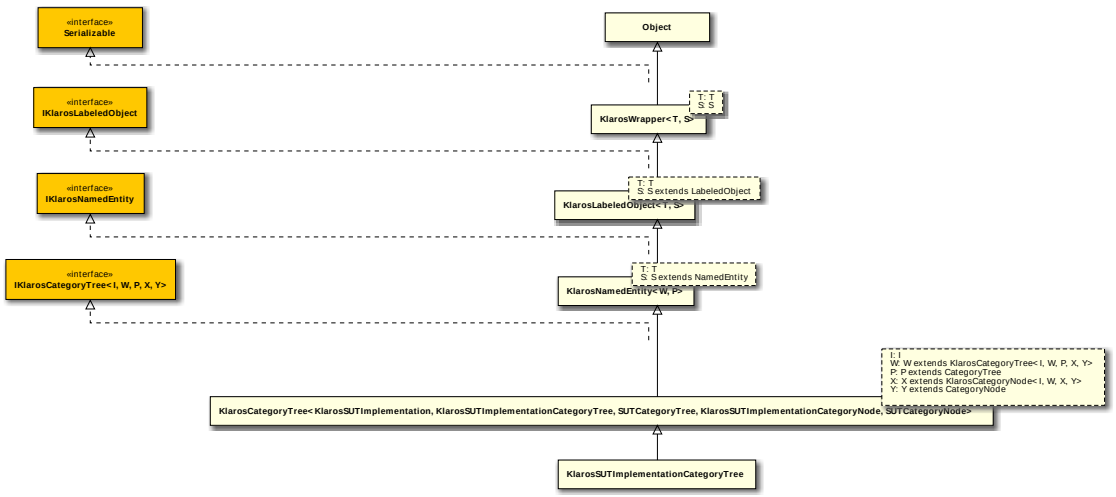
Methods inherited from de.verit.klaros.core.model.KlarosCategoryTree: `getDescription` , `getRootNode` , `getShortname`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.56. KlarosTag

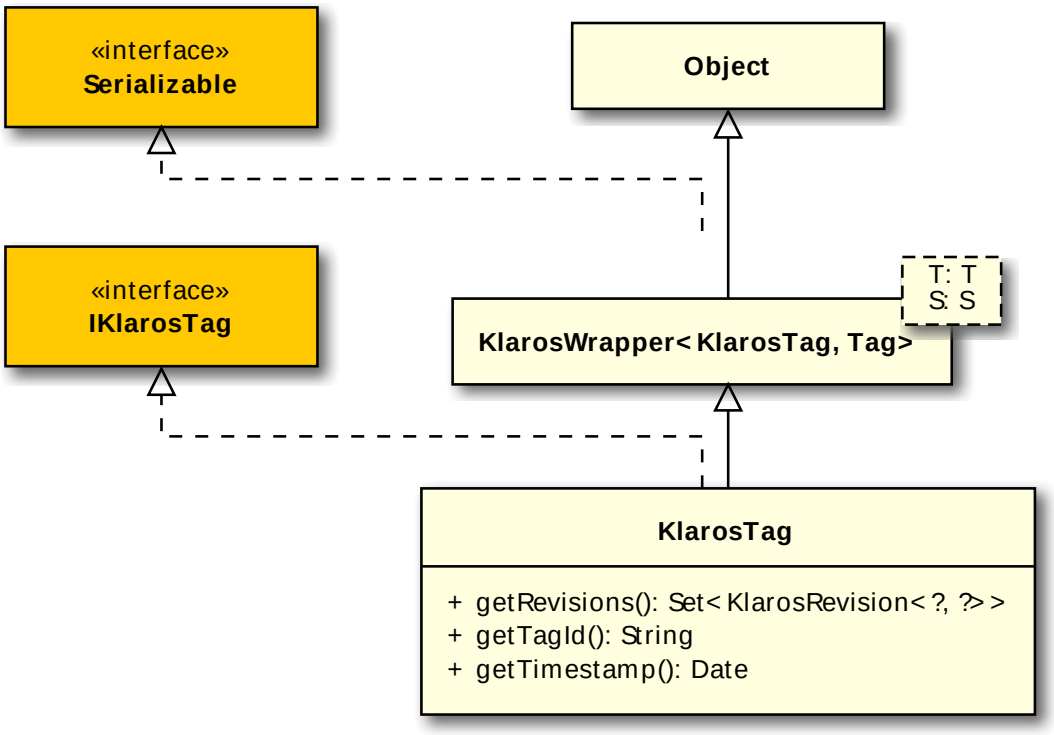
This class encapsulates the tag information of a taggable klaros object.

B.1.1.56.1. Synopsis

```
public final class KlarosTag extends KlarosWrapper<KlarosTag, Tag>
   implements IKlarosTag {

  // Public Methods
  public Set<KlarosRevision<?, ?>> getRevisions();
  public String getTagId();
  public Date getTimestamp();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.57. KlarosTestCase

This class provides access to the information stored for a test case.

B.1.1.57.1. Synopsis

```
public final class KlarosTestCase extends
                                KlarosRevision<KlarosTestCase, TestCase>

  implements IKlarosTestCase<KlarosTestCase, TestCase>,
          Comparable<KlarosTestCase> {

 // Public Methods
 public int compareTo(KlarosTestCase o);
 public boolean equals(Object o);
 public TestAreatopic getArea();
 public Set<KlarosAttachment> getAttachments();
 public KlarosTestCase getBranchRoot();
 public Set<KlarosRevision<KlarosTestCase, TestCase>> getBranches();
 public Set<KlarosTestCaseCategoryNode> getCategories();
 public KlarosConfiguration getConfiguration();
 public Set<KlarosRequirement> getCovers();
 public String getDependency();
 public String getDescription();
 public TestDesignTechnique getDesignTechnique();
 public List<KlarosIssue> getDetectedIssues();
 public String getDocbase();
 public String getEvaluation();
 public TestExecutionMethod getExecution();
```

```
  public String getExpectedResult();
  public Set<String> getExternalNames();
  public KlarosExternalImplementation getImplementation();
  public TestLevel getLevel();
  public String getNote();
  public String getPostcondition();
  public String getPrecondition();
  public KlarosTestCase getPredecessor();
  public TestPriority getPriority();
  public Set<KlarosTestCaseResult> getResults();
  public KlarosTestCase getRoot();
  public String getShortname();
  public String getState();
  public String getStateName(KlarosSUTImplementation sut);
  public Map<KlarosSUTImplementation, KlarosTestCaseState> getStates();
  public KlarosTestCase getSuccessor();
  public String getTeam();
  public List<KlarosTestCaseStep> getTestCaseSteps();
  public String getTraceability();
  public KlarosTestCase getTrunkRoot();
  public TestVariety getVariety();
  public int hashCode();
}
```

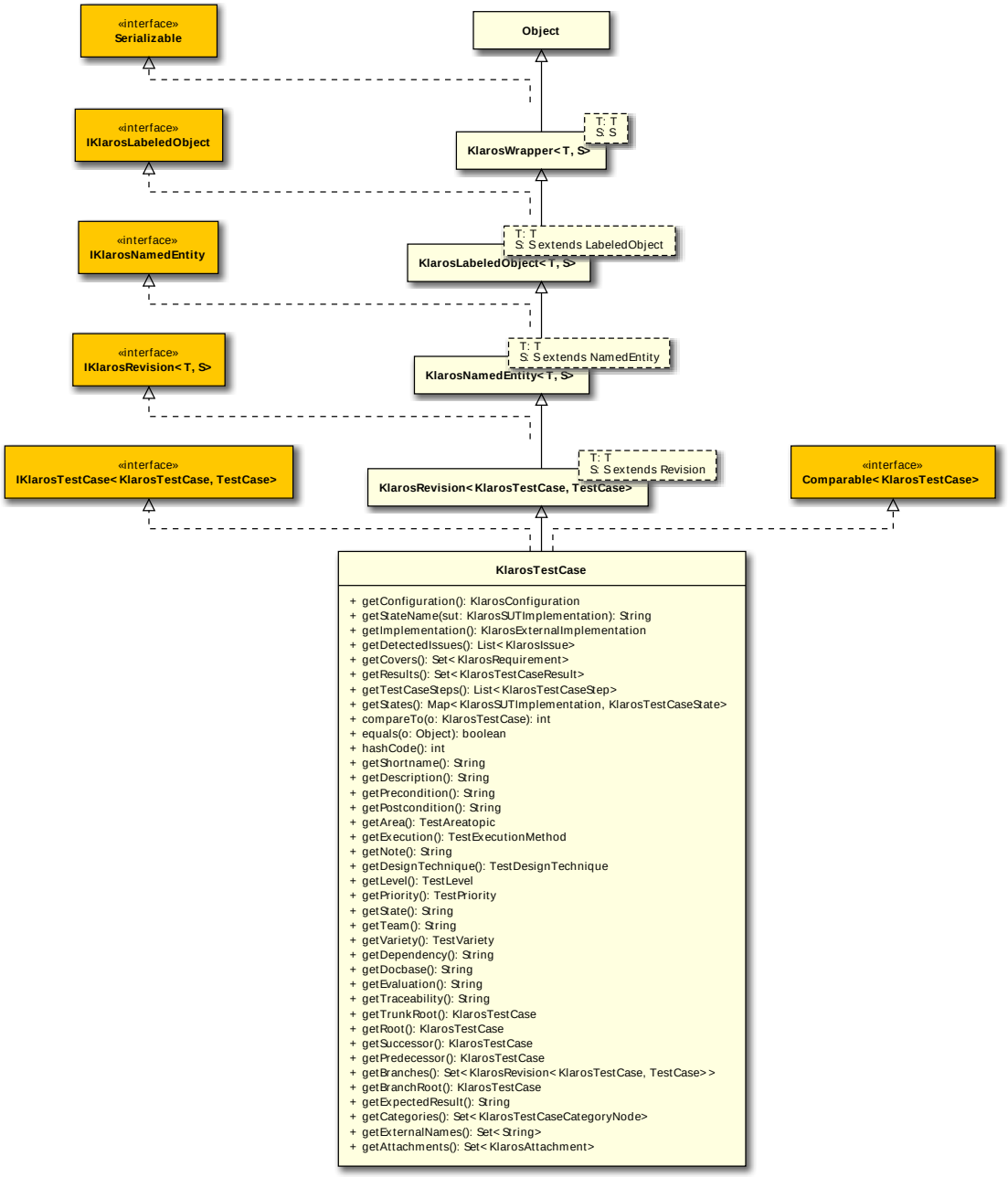
Methods inherited from de.verit.klaros.core.model.KlarosRevision: getRevisionComment , getRevisionId

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: getAssignee , getProperties , isDefinedProperty

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: getWrapped

Methods inherited from java.lang.Object: equals , getClass , hashCode , notify , notifyAll , toString , wait



B.1.1.58. KlarosTestCaseCategoryNode

A category node for test cases.

B.1.1.58.1. Synopsis

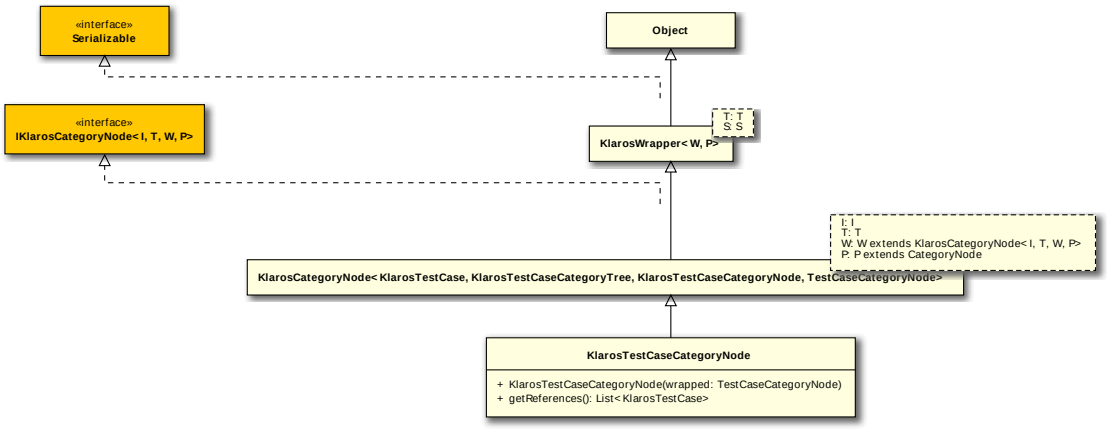
```
public final class KlarosTestCaseCategoryNode extends
                                KlarosCategoryNode<KlarosTestCase,
 KlarosTestCaseCategoryTree, KlarosTestCaseCategoryNode, TestCaseCategoryNode>
                               {

  // Public Constructors
  public KlarosTestCaseCategoryNode(TestCaseCategoryNode wrapped);

  // Public Methods
```

```
  public List<KlarosTestCase> getReferences();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosCategoryNode: `getCategoryTree` , `getDescription` , `getParent` , `getShortname` , `getSubCategories`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.58.2. KlarosTestCaseCategoryNode(TestCaseCategoryNode)

```
  public KlarosTestCaseCategoryNode(TestCaseCategoryNode wrapped);
```

Instantiates a new klaros test case category node.

Parameters

`wrapped`	the wrapped class

B.1.1.59. KlarosTestCaseCategoryTree

A category node for test cases.

B.1.1.59.1. Synopsis

```
public final class KlarosTestCaseCategoryTree extends
                                KlarosCategoryTree<KlarosTestCase,
 KlarosTestCaseCategoryTree, TestCaseCategoryTree, KlarosTestCaseCategoryNode,
 TestCaseCategoryNode>
                              {
}
```

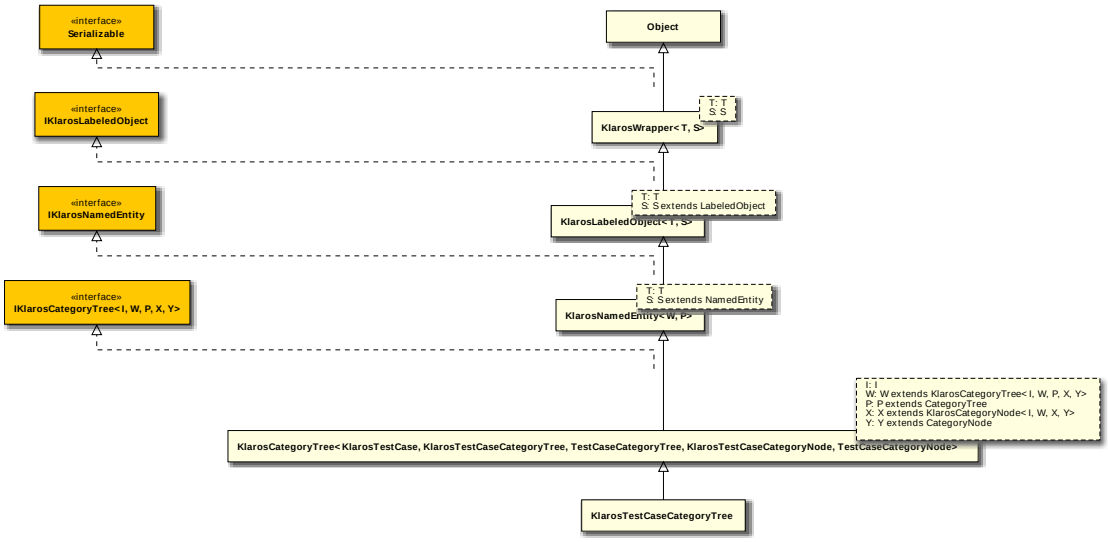
Methods inherited from de.verit.klaros.core.model.KlarosCategoryTree: `getDescription` , `getRootNode` , `getShortname`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.60. KlarosTestCaseResult

This class provides access to the information stored for test case results.

B.1.1.60.1. Synopsis

```
public final class KlarosTestCaseResult extends
                                KlarosNamedEntity<KlarosTestCaseResult,
 TestCaseResult>

  implements IKlarosTestCaseResult,
          Comparable<KlarosTestCaseResult> {

 // Public Methods
 public int compareTo(KlarosTestCaseResult o);
 public boolean equals(Object o);
 public Set<KlarosAttachment> getAttachments();
 public String getDescription();
 public long getExecutionTime();
 public List<KlarosTestCaseStepResult> getStepResults();
 public String getSummary();
 public KlarosTestCase getTestCase();
 public KlarosTestRun getTestRun();
 public Integer getTestSuitePosition();
 public KlarosTestSuiteResult getTestSuiteResult();
 public int hashCode();
 public boolean isError();
 public boolean isFailure();
```

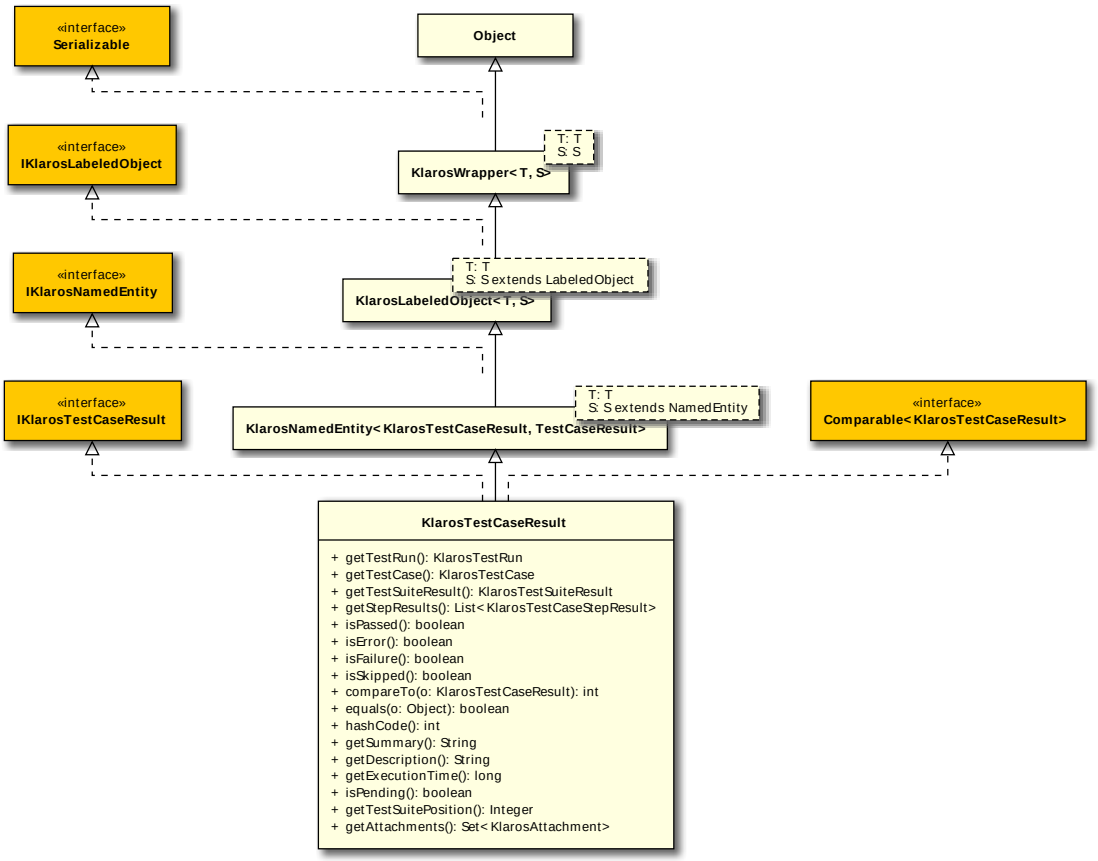
```
  public boolean isPassed();
  public boolean isPending();
  public boolean isSkipped();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.61. KlarosTestCaseState

This class provides access to the information stored for test case states.

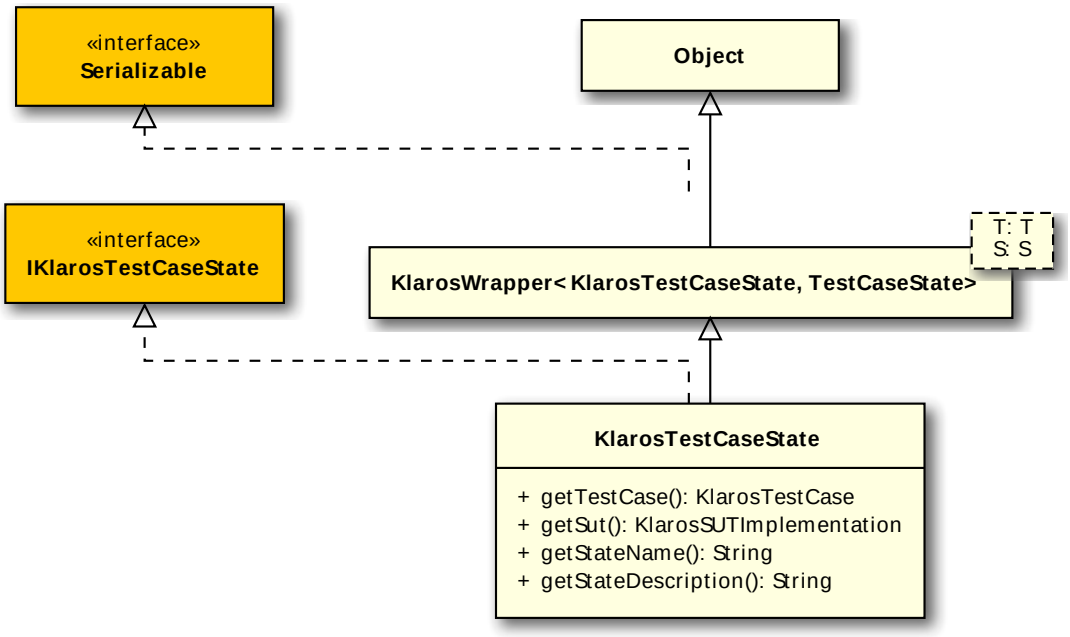
B.1.1.61.1. Synopsis

```
public final class KlarosTestCaseState extends KlarosWrapper<KlarosTestCaseState,
 TestCaseState>
  implements IKlarosTestCaseState {

 // Public Methods
```

```
  public String getStateDescription();
  public String getStateName();
  public KlarosSUTImplementation getSut();
  public KlarosTestCase getTestCase();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.62. KlarosTestCaseStep

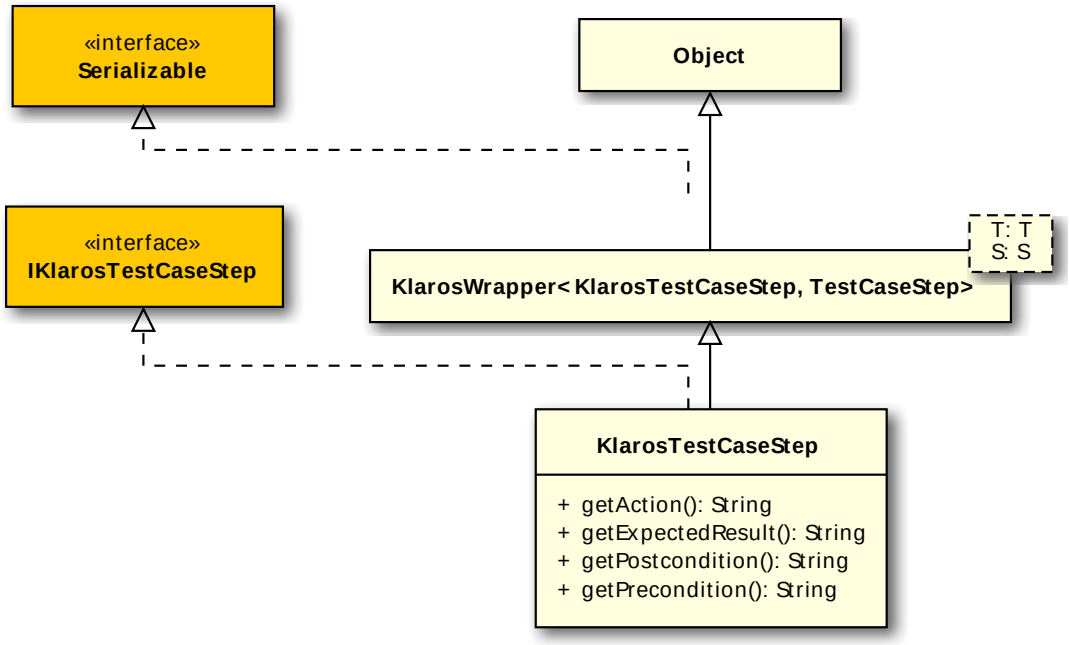
This class provides access to the information stored for test case steps.

B.1.1.62.1. Synopsis

```
public final class KlarosTestCaseStep extends KlarosWrapper<KlarosTestCaseStep,
 TestCaseStep>
  implements IKlarosTestCaseStep {

  // Public Methods
  public String getAction();
  public String getExpectedResult();
  public String getPostcondition();
  public String getPrecondition();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.63. KlarosTestCaseStepResult

This class provides access to the information stored for test case step results.

B.1.1.63.1. Synopsis

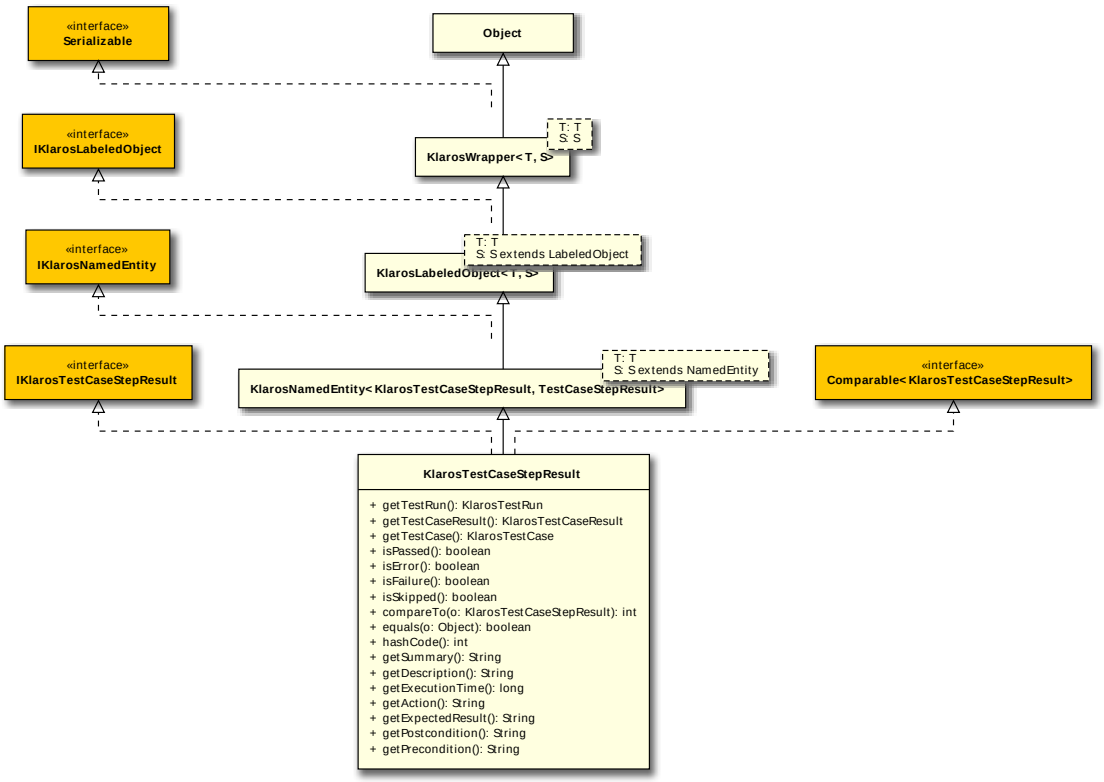
```
public final class KlarosTestCaseStepResult extends
                                KlarosNamedEntity<KlarosTestCaseStepResult,
 TestCaseStepResult>

  implements IKlarosTestCaseStepResult,
          Comparable<KlarosTestCaseStepResult> {

 // Public Methods
 public int compareTo(KlarosTestCaseStepResult o);
 public boolean equals(Object o);
 public String getAction();
 public String getDescription();
 public long getExecutionTime();
 public String getExpectedResult();
 public String getPostcondition();
 public String getPrecondition();
 public String getSummary();
 public KlarosTestCase getTestCase();
 public KlarosTestCaseResult getTestCaseResult();
 public KlarosTestRun getTestRun();
 public int hashCode();
 public boolean isError();
 public boolean isFailure();
 public boolean isPassed();
 public boolean isSkipped();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.64. KlarosTestEnvironment

This class provides access to the information stored for test environments.

B.1.1.64.1. Synopsis

```
public final class KlarosTestEnvironment extends
                                KlarosNamedEntity<KlarosTestEnvironment,
 TestEnvironment>

  implements IKlarosTestEnvironment,
         Comparable<KlarosTestEnvironment> {

 // Public Methods
 public int compareTo(KlarosTestEnvironment o);
 public boolean equals(Object o);
 public Set<KlarosAttachment> getAttachments();
 public Set<KlarosTestEnvironmentCategoryNode> getCategories();
```

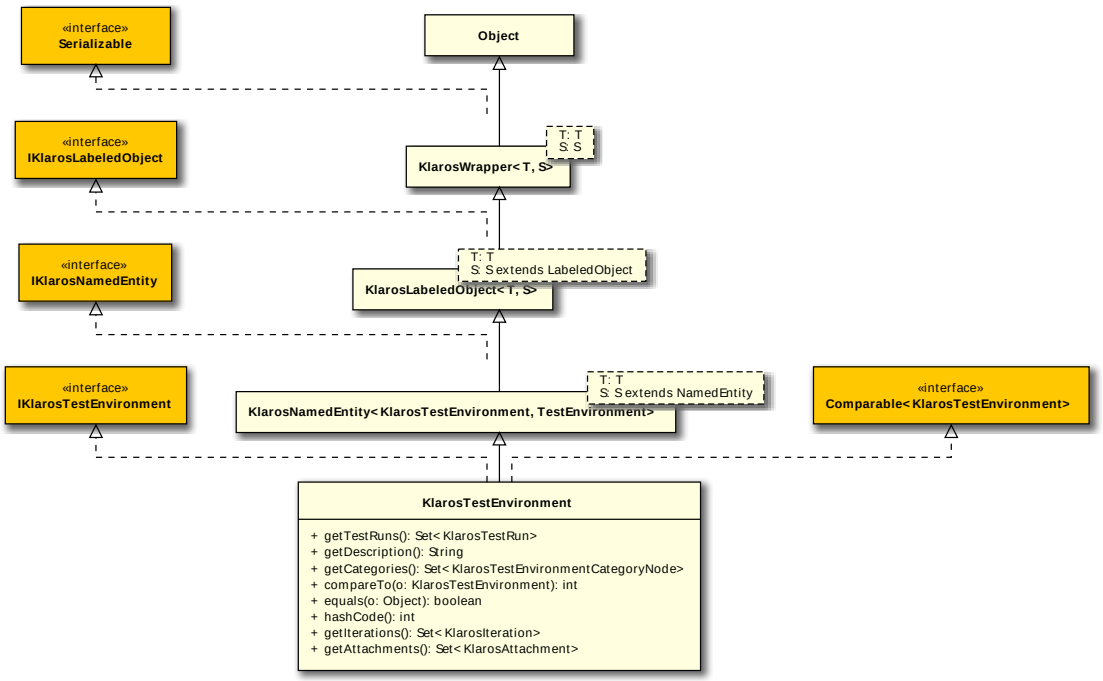
```
  public String getDescription();
  public Set<KlarosIteration> getIterations();
  public Set<KlarosTestRun> getTestRuns();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.65. KlarosTestEnvironmentCategoryNode

A category node for test environments.

B.1.1.65.1. Synopsis

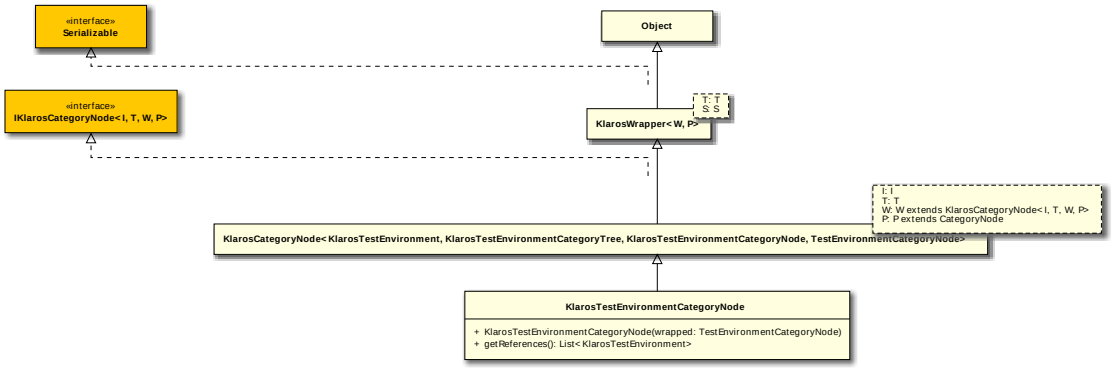
```
public final class KlarosTestEnvironmentCategoryNode extends
                              KlarosCategoryNode<KlarosTestEnvironment,
 KlarosTestEnvironmentCategoryTree, KlarosTestEnvironmentCategoryNode,
 TestEnvironmentCategoryNode>
                             {

  // Public Constructors
  public KlarosTestEnvironmentCategoryNode(TestEnvironmentCategoryNode wrapped);
```

```
  // Public Methods
  public List<KlarosTestEnvironment> getReferences();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosCategoryNode: getCategoryTree , getDescription , getParent , getShortname , getSubCategories

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: getWrapped

Methods inherited from java.lang.Object: equals , getClass , hashCode , notify , notifyAll , toString , wait



B.1.1.65.2. KlarosTestEnvironmentCategoryNode(TestEnvironmentCategoryNode)

```
public KlarosTestEnvironmentCategoryNode(TestEnvironmentCategoryNode wrapped);
```

Instantiates a new klaros test environment category node.

Parameters

wrapped	the wrapped class

B.1.1.66. KlarosTestEnvironmentCategoryTree

A category node for test environments.

B.1.1.66.1. Synopsis

```
public final class KlarosTestEnvironmentCategoryTree extends
                                KlarosCategoryTree<KlarosTestEnvironment,
 KlarosTestEnvironmentCategoryTree, TestEnvironmentCategoryTree,
 KlarosTestEnvironmentCategoryNode, TestEnvironmentCategoryNode>
                               {
}
```

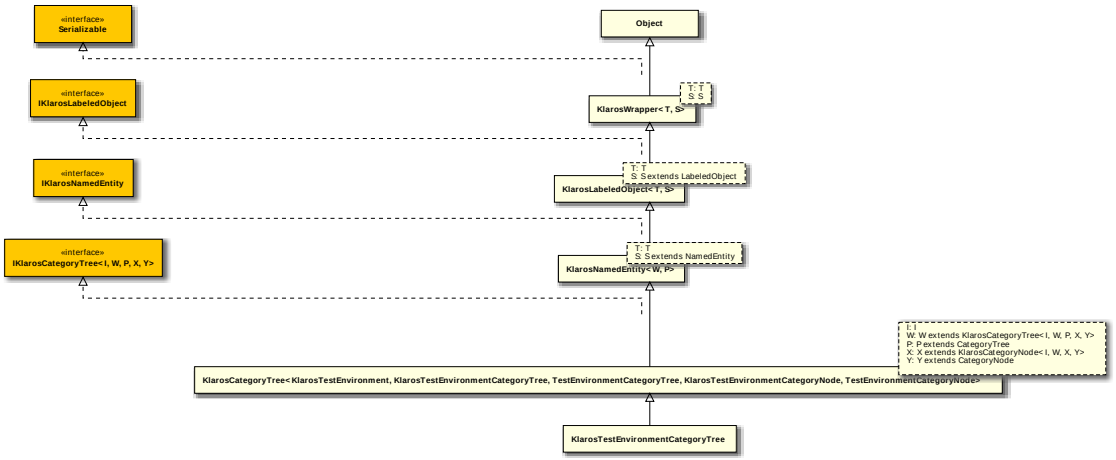
Methods inherited from de.verit.klaros.core.model.KlarosCategoryTree: getDescription , getRootNode , getShortname

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: getAssignee , getProperties , isDefinedProperty

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.67. KlarosTestExecutable

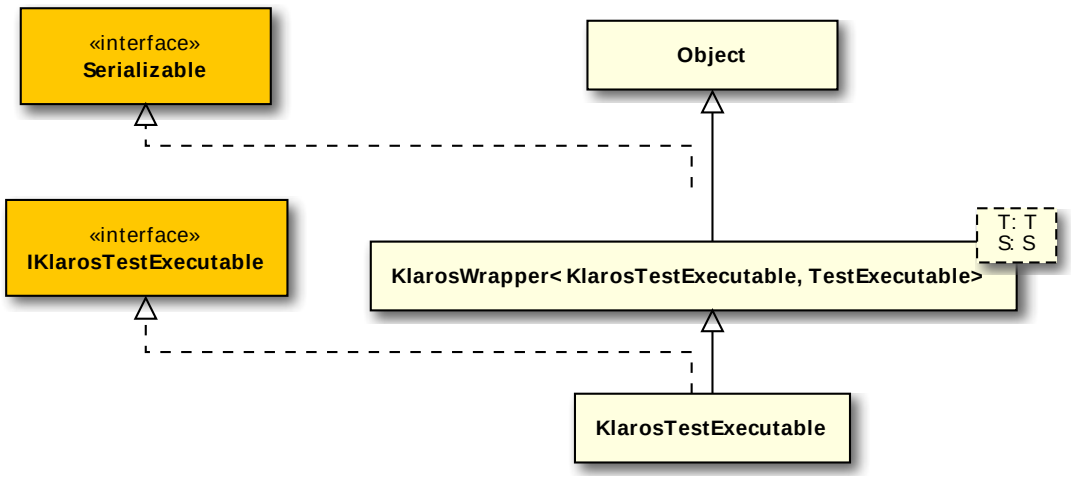
This class provides access to the information stored for test cases.

B.1.1.67.1. Synopsis

```
public final class KlarosTestExecutable extends KlarosWrapper<KlarosTestExecutable,
 TestExecutable>
   implements IKlarosTestExecutable {
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.68. KlarosTestRun

This class provides access to the information stored for test runs.

B.1.1.68.1. Synopsis

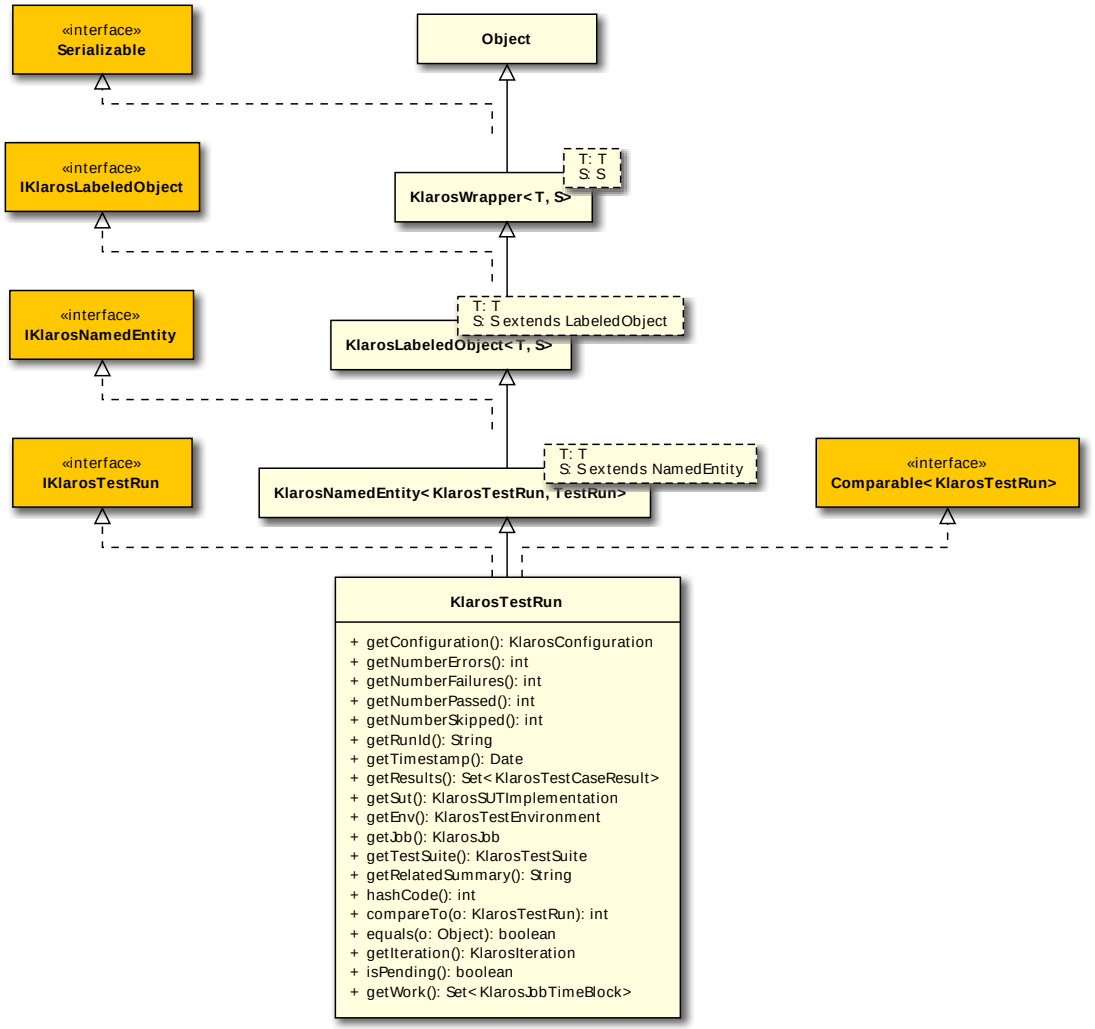
```
public final class KlarosTestRun extends
                                KlarosNamedEntity<KlarosTestRun, TestRun>

   implements IKlarosTestRun,
           Comparable<KlarosTestRun> {

  // Public Methods
  public int compareTo(KlarosTestRun o);
  public boolean equals(Object o);
  public KlarosConfiguration getConfiguration();
  public KlarosTestEnvironment getEnv();
  public KlarosIteration getIteration();
  public KlarosJob getJob();
  public int getNumberErrors();
  public int getNumberFailures();
  public int getNumberPassed();
  public int getNumberSkipped();
  public String getRelatedSummary();
  public Set<KlarosTestCaseResult> getResults();
  public String getRunId();
  public KlarosSUTImplementation getSut();
  public KlarosTestSuite getTestSuite();
  public Date getTimestamp();
  public Set<KlarosJobTimeBlock> getWork();
  public int hashCode();
  public boolean isPending();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.69. KlarosTestSuite

This class provides access to the information stored for test suites.

B.1.1.69.1. Synopsis

```
public final class KlarosTestSuite extends
                              KlarosRevision<KlarosTestSuite, TestSuite>

  implements IKlarosTestSuite<KlarosTestSuite, TestSuite> {

 // Public Methods
 public Set<KlarosAttachment> getAttachments();
 public KlarosTestSuite getBranchRoot();
 public Set<KlarosRevision<KlarosTestSuite, TestSuite>> getBranches();
 public Set<KlarosTestSuiteCategoryNode> getCategories();
 public KlarosConfiguration getConfiguration();
 public KlarosTestSuite getPredecessor();
 public List<KlarosTestSuiteResult> getResults();
 public KlarosTestSuite getRoot();
 public String getShortname();
 public KlarosTestSuite getSuccessor();
```

```
  public KlarosSUTImplementation getSut();
  public List<KlarosTestCase> getTestCases();
  public Integer getTestSuiteResultCount();
  public KlarosTestSuite getTrunkRoot();
}
```

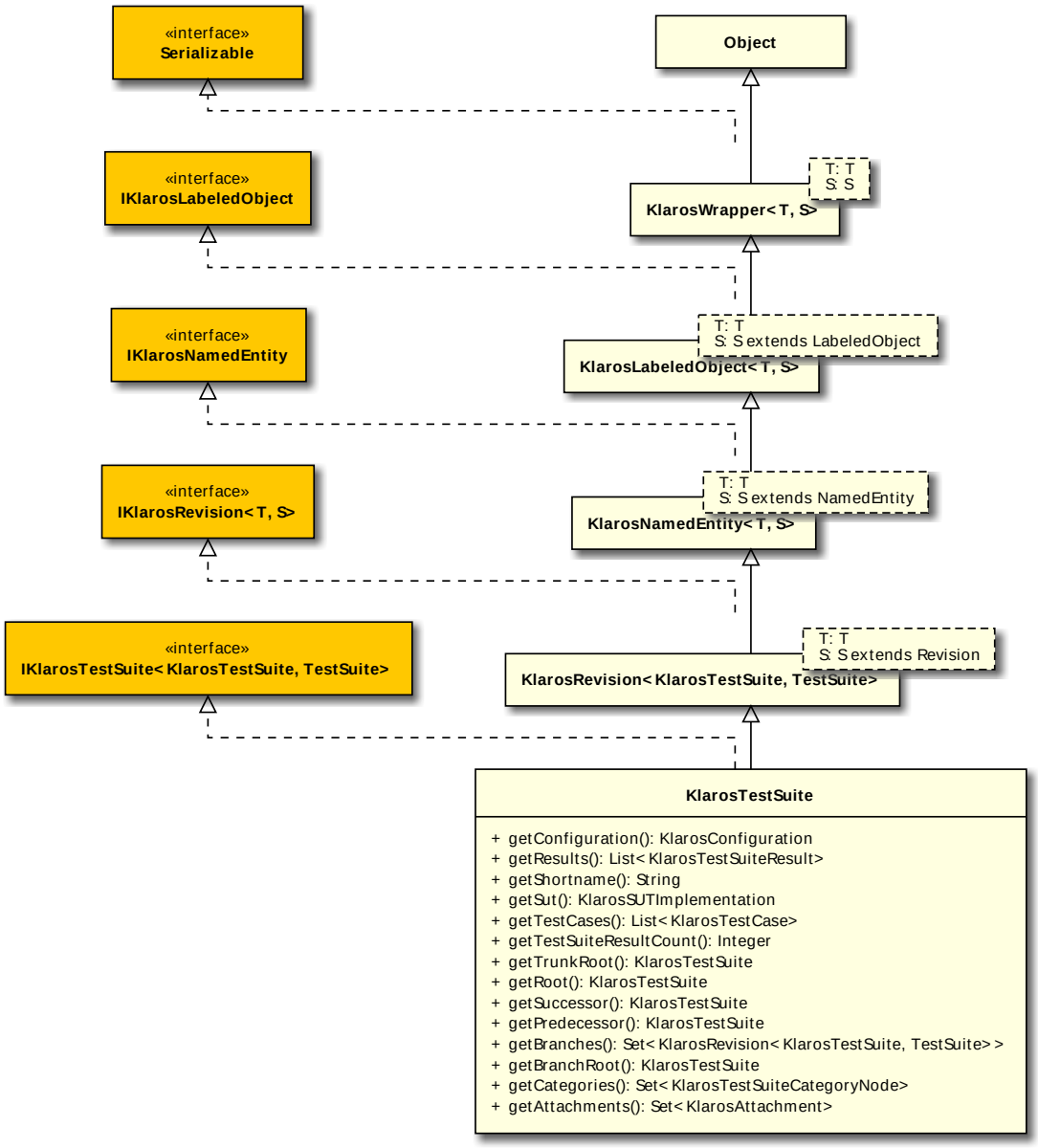
Methods inherited from de.verit.klaros.core.model.KlarosRevision: `getRevisionComment` , `getRevisionId`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.70. KlarosTestSuiteCategoryNode

A category node for test suites.

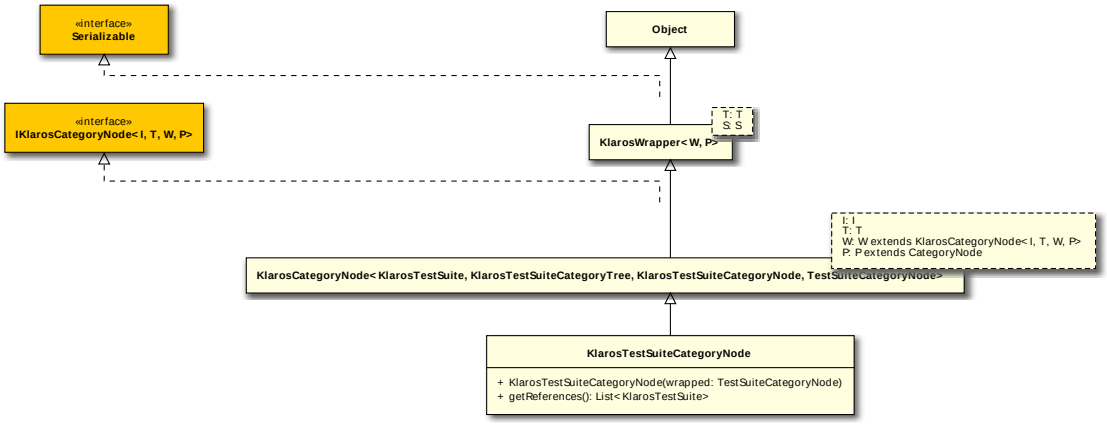
B.1.1.70.1. Synopsis

```
public final class KlarosTestSuiteCategoryNode extends
                                KlarosCategoryNode<KlarosTestSuite,
 KlarosTestSuiteCategoryTree, KlarosTestSuiteCategoryNode, TestSuiteCategoryNode>
                               {

  // Public Constructors
  public KlarosTestSuiteCategoryNode(TestSuiteCategoryNode wrapped);

  // Public Methods
  public List<KlarosTestSuite> getReferences();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosCategoryNode: `getCategoryTree` , `getDescription` , `getParent` , `getShortname` , `getSubCategories`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.70.2. KlarosTestSuiteCategoryNode(TestSuiteCategoryNode)

```
public KlarosTestSuiteCategoryNode(TestSuiteCategoryNode wrapped);
```

Instantiates a new klaros test suite category node.

Parameters

`wrapped`	the wrapped class

B.1.1.71. KlarosTestSuiteCategoryTree

A category node for test suites.

B.1.1.71.1. Synopsis

```
public final class KlarosTestSuiteCategoryTree extends
                                KlarosCategoryTree<KlarosTestSuite,
 KlarosTestSuiteCategoryTree, TestSuiteCategoryTree, KlarosTestSuiteCategoryNode,
 TestSuiteCategoryNode>
                               {
}
```

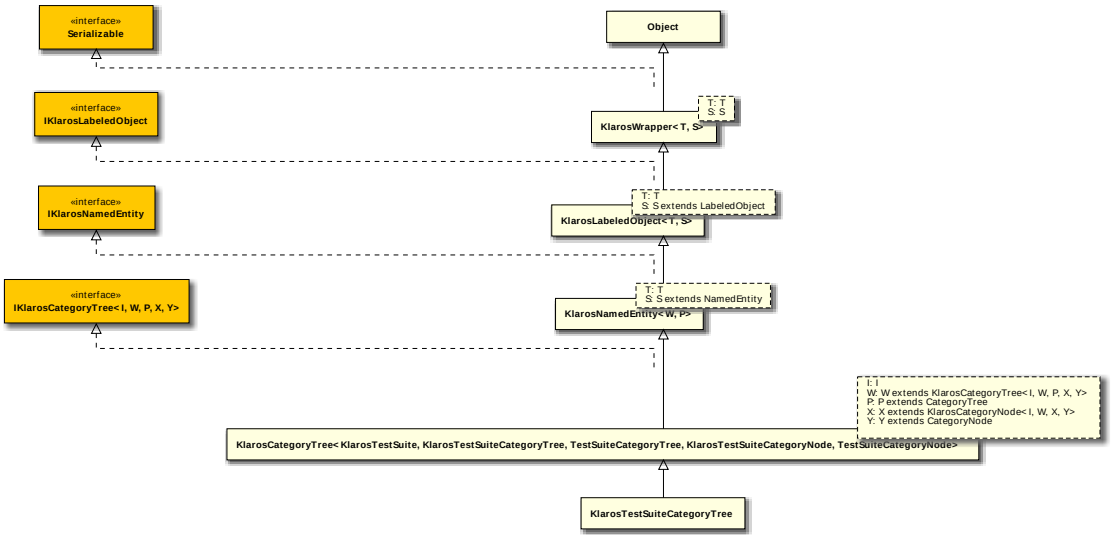
Methods inherited from de.verit.klaros.core.model.KlarosCategoryTree: `getDescription` , `getRootNode` , `getShortname`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.72. KlarosTestSuiteResult

This class provides access to the information stored for test suite results.

B.1.1.72.1. Synopsis

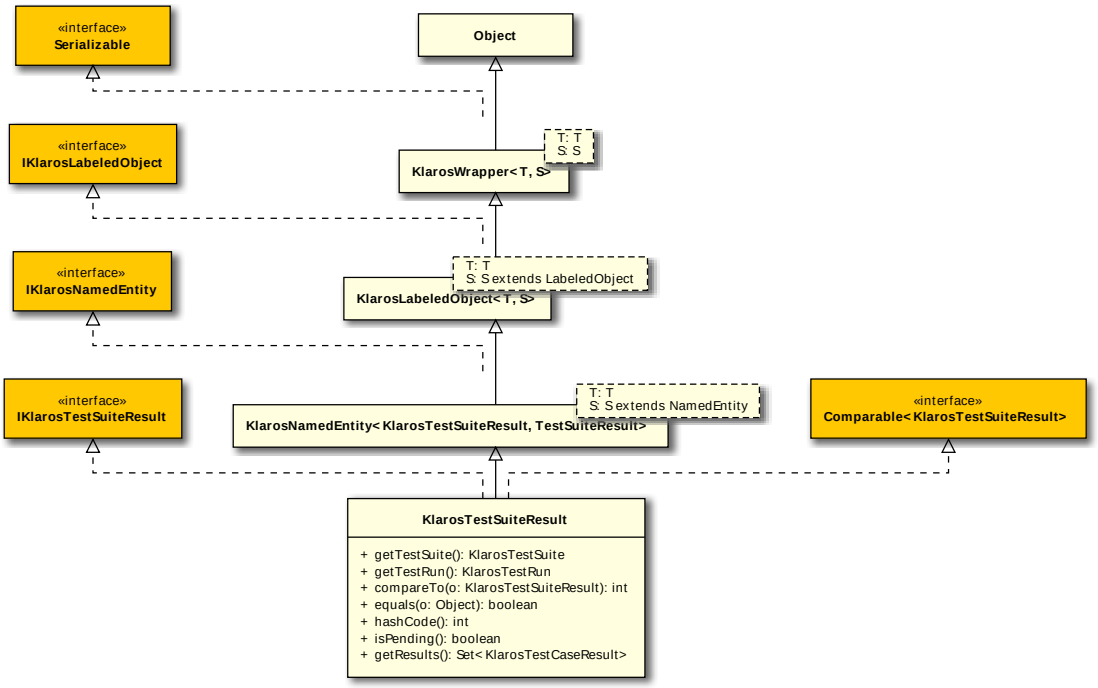
```
public final class KlarosTestSuiteResult extends
                                KlarosNamedEntity<KlarosTestSuiteResult,
 TestSuiteResult>

  implements IKlarosTestSuiteResult,
          Comparable<KlarosTestSuiteResult> {

  // Public Methods
  public int compareTo(KlarosTestSuiteResult o);
  public boolean equals(Object o);
  public Set<KlarosTestCaseResult> getResults();
  public KlarosTestRun getTestRun();
  public KlarosTestSuite getTestSuite();
  public int hashCode();
  public boolean isPending();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.73. KlarosUser

The user object.

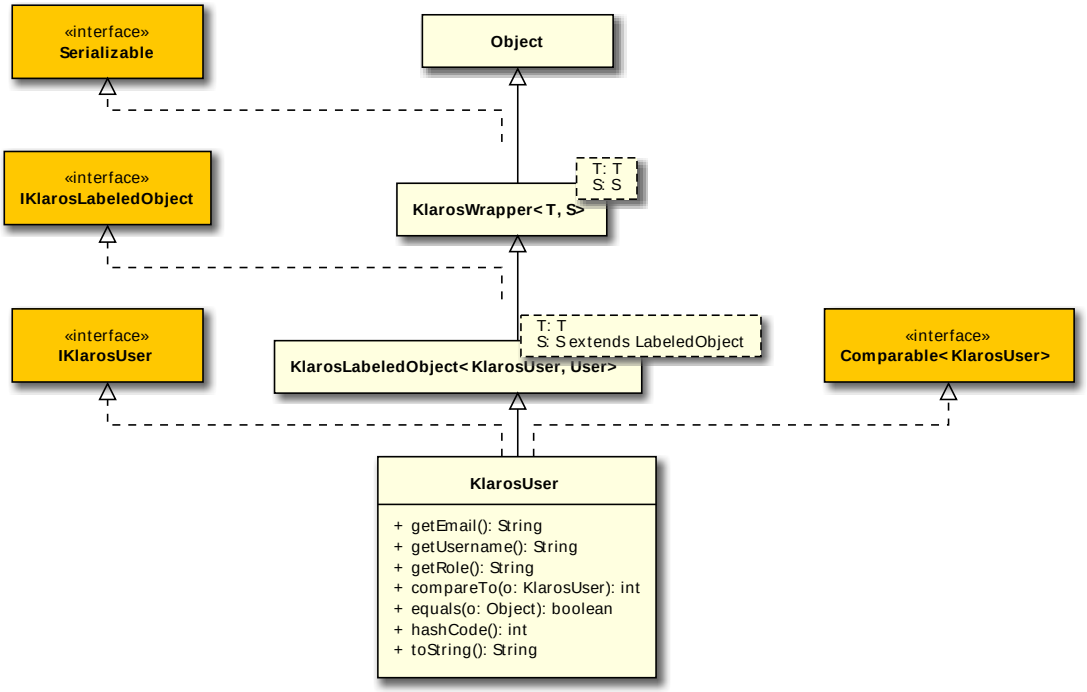
B.1.1.73.1. Synopsis

```
public class KlarosUser extends
                                KlarosLabeledObject<KlarosUser, User>

  implements IKlarosUser,
         Comparable<KlarosUser> {

  // Public Methods
  public int compareTo(KlarosUser o);
  public boolean equals(Object o);
  public String getEmail();
  public String getRole();
  public String getUsername();
  public int hashCode();
  public String toString();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `getWrapped`

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.2. Scripting API-Reference

B.2.1. de.verit.klaros.scripting

B.2.1.1. KlarosContext

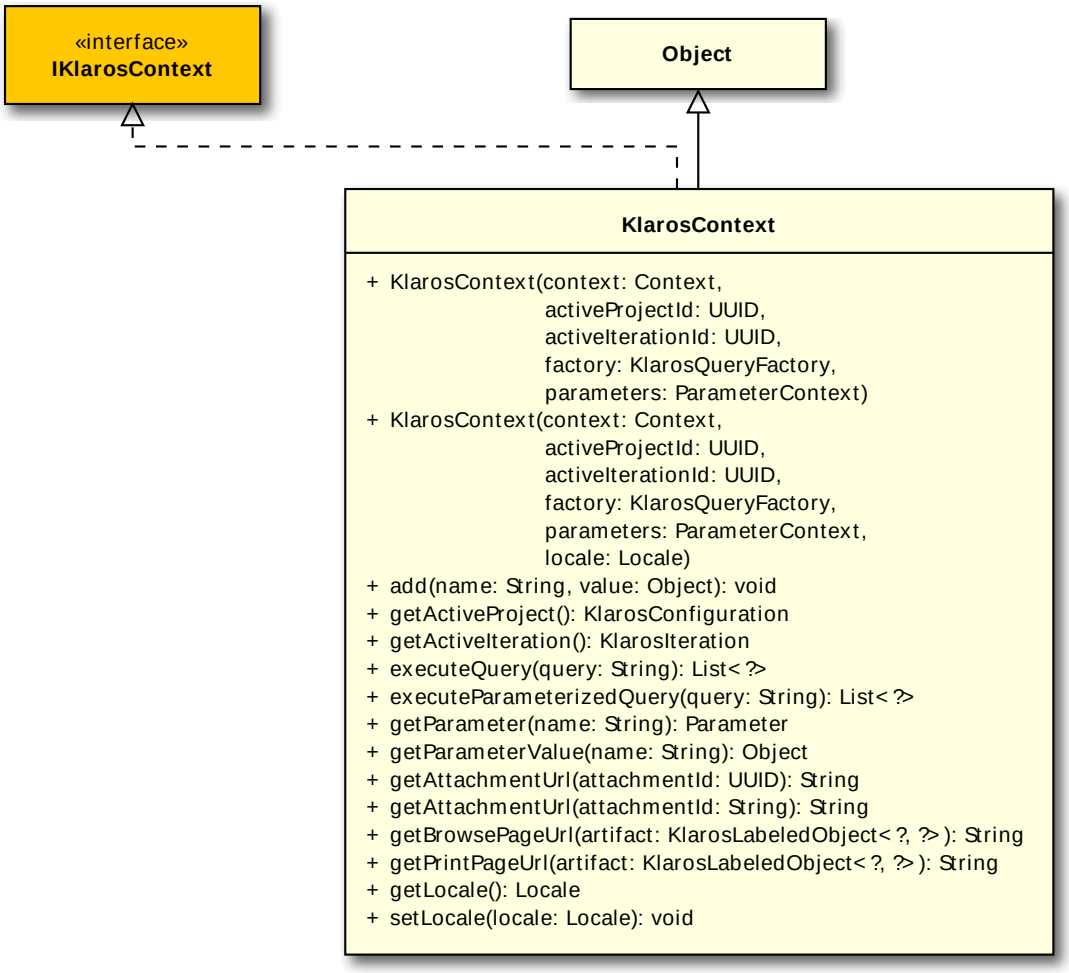
Context to provide all methods to the user to add own objects to the event context.

B.2.1.1.1. Synopsis

```
public class KlarosContext implements IKlarosContext {

  // Public Constructors
  public KlarosContext(Context context,
                  UUID activeProjectId,
                  UUID activeIterationId,
                  KlarosQueryFactory factory,
                  ParameterContext parameters);
  public KlarosContext(Context context,
                  UUID activeProjectId,
                  UUID activeIterationId,
                  KlarosQueryFactory factory,
                  ParameterContext parameters,
                  Locale locale);

  // Public Methods
  public void add(String name, Object value);
  public List<?> executeParameterizedQuery(String query);
  public List<?> executeQuery(String query);
  public KlarosIteration getActiveIteration();
  public KlarosConfiguration getActiveProject();
```

```
  public String getAttachmentUrl(String attachmentId);
  public String getAttachmentUrl(UUID attachmentId);
  public String getBrowsePageUrl(KlarosLabeledObject<?, ?> artifact);
  public Locale getLocale();
  public Parameter getParameter(String name);
  public Object getParameterValue(String name);
  public String getPrintPageUrl(KlarosLabeledObject<?, ?> artifact);
  public void setLocale(Locale locale);
}
```

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.2.1.1.2. KlarosContext(Context, UUID, UUID, KlarosQueryFactory, ParameterContext)

```
  public KlarosContext(Context context,
                       UUID activeProjectId,
                       UUID activeIterationId,
                       KlarosQueryFactory factory,
                       ParameterContext parameters);
```

Create a KlarosContext.

Parameters

`context`	Gets passed by the relating servlet.
`activeProjectId`	the active project id
`activeIterationId`	the active iteration id if available
`factory`	the factory
`parameters`	the parameters

B.2.1.1.3. KlarosContext(Context, UUID, UUID, KlarosQueryFactory, ParameterContext, Locale)

```
public KlarosContext(Context context,
                UUID activeProjectId,
                UUID activeIterationId,
                KlarosQueryFactory factory,
                ParameterContext parameters,
                Locale locale);
```

Create a KlarosContext.

Parameters

`context`	Gets passed by the relating servlet.
`activeProjectId`	the active project id
`activeIterationId`	the active iteration id
`factory`	the factory
`parameters`	the parameters
`locale`	the locale

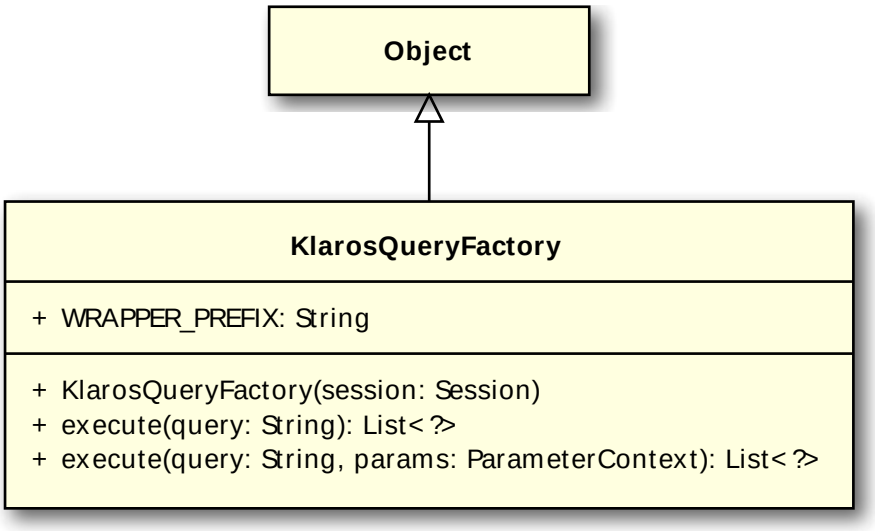
B.2.1.2. KlarosQueryFactory

This factory lets the user execute queries.

B.2.1.2.1. Synopsis

```
public class KlarosQueryFactory {

  // Public Static Fields
  public static final String WRAPPER_PREFIX = "Klaros";

  // Public Constructors
  public KlarosQueryFactory(Session session);

  // Public Methods
  public List<?> execute(String query);
  public List<?> execute(String query, ParameterContext params);
}
```

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.2.1.2.2. KlarosQueryFactory(Session)

```
public KlarosQueryFactory(Session session);
```

Create a KlarosQueryFactory.

Parameters

`session`	Passed by the relating servlet.

B.2.1.2.3. WRAPPER_PREFIX

```
public static final String WRAPPER_PREFIX = "Klaros";
```

The prefix for klaros interfaces.

B.2.1.2.4. execute(String)

```
public List<?> execute(String query);
```

Prepares the given query string with our KlarosOdaQuery and returns a KlarosList with the results of the query.

Parameters

`query`	the query
`return`	the list of query results

B.2.1.2.5. execute(String, ParameterContext)

```
public List<?> execute(String query, ParameterContext params);
```

Prepares the given query string with our KlarosOdaQuery and returns a KlarosList with the results of the query.

Parameters

`query`	the query
`params`	the parameters

`return`	the list<?>

B.2.1.3. KlarosScript

Public interface that all seam-pdf template scripts must implement be work properly.

B.2.1.3.1. Synopsis

```
public interface KlarosScript {

  // Public Methods
  public abstract void execute(KlarosContext context);
}
```

B.2.1.3.2. execute(KlarosContext)

```
public abstract void execute(KlarosContext context);
```

This functions gets called by the seam-pdf servlet to execute the script.

Parameters

`context`	The event context to provide all needed functions, properties and objects.

B.2.2. de.verit.klaros.scripting.context

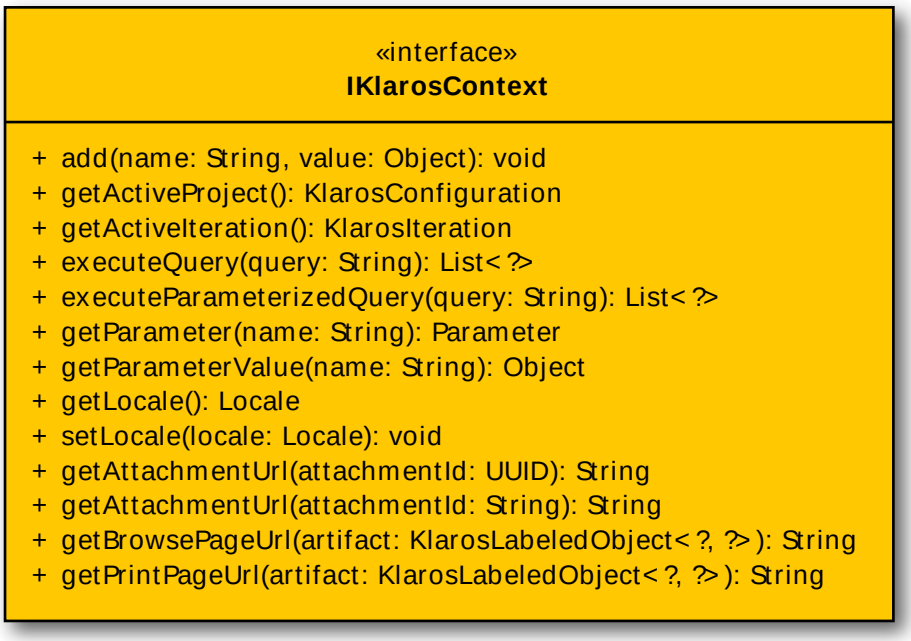
B.2.2.1. IKlarosContext

Describes the available actions in the scripting context.

B.2.2.1.1. Synopsis

```
public interface IKlarosContext {

  // Public Methods
  public abstract void add(String name, Object value);
  public abstract List<?> executeParameterizedQuery(String query);
  public abstract List<?> executeQuery(String query);
  public abstract KlarosIteration getActiveIteration();
  public abstract KlarosConfiguration getActiveProject();
  public abstract String getAttachmentUrl(String attachmentId);
  public abstract String getAttachmentUrl(UUID attachmentId);
  public abstract String getBrowsePageUrl(KlarosLabeledObject<?, ?> artifact);
  public abstract Locale getLocale();
  public abstract Parameter getParameter(String name);
  public abstract Object getParameterValue(String name);
```

```
  public abstract String getPrintPageUrl(KlarosLabeledObject<?, ?> artifact);
  public abstract void setLocale(Locale locale);
}
```



B.2.2.1.2. add(String, Object)

```
public abstract void add(String name, Object value);
```

Add a new object with the give key to the event context.

Parameters

`name`	the name
`value`	the value

B.2.2.1.3. executeParameterizedQuery(String)

```
public abstract List<?> executeParameterizedQuery(String query);
```

Execute the given query with the Parameters from the ParameterContext.

Parameters

`query`	The HQL query to execute.
`return`	A KlarosList of the selected objects.

B.2.2.1.4. executeQuery(String)

```
public abstract List<?> executeQuery(String query);
```

Execute the given query.

Parameters

`query`	The HQL query to execute.

`return`	A KlarosList of the selected objects.

B.2.2.1.5. getActiveIteration()

```
public abstract KlarosIteration getActiveIteration();
```

Gets the active iteration.

Parameters

`return`	the active iteration

B.2.2.1.6. getActiveProject()

```
public abstract KlarosConfiguration getActiveProject();
```

Gets the active project.

Parameters

`return`	the active project

B.2.2.1.7. getAttachmentUrl(String)

```
public abstract String getAttachmentUrl(String attachmentId);
```

Return the attachment url for the given attachment id.

Parameters

`attachmentId`	the attachment id
`return`	the attachment url

B.2.2.1.8. getAttachmentUrl(UUID)

```
public abstract String getAttachmentUrl(UUID attachmentId);
```

Return the attachment url for the given attachment id.

Parameters

`attachmentId`	the attachment id
`return`	the attachment url

B.2.2.1.9. getBrowsePageUrl(KlarosLabeledObject<?, ?>)

```
public abstract String getBrowsePageUrl(KlarosLabeledObject<?, ?> artifact);
```

Gets the browse page url to the given klaros artifact. Will return null if the artifact type is not supported.

Parameters

`artifact`	the artifact
`return`	the browse page url

B.2.2.1.10. getParameter(String)

```
public abstract Parameter getParameter(String name);
```

Gets the parameter.

Parameters

`name`	the name
`return`	The parameter with the given name or `null`.

B.2.2.1.11. getParameterValue(String)

```
public abstract Object getParameterValue(String name);
```

Gets the parameter value.

Parameters

`name`	the name
`return`	The value of the parameter with the given name or `null`.

B.2.2.1.12. getPrintPageUrl(KlarosLabeledObject<?, ?>)

```
public abstract String getPrintPageUrl(KlarosLabeledObject<?, ?> artifact);
```

Gets the print page of the given klaros artifact. Will return null if the artifact type is not supported.

Parameters

`artifact`	the artifact
`return`	the print page url

B.2.2.1.13. setLocale(Locale)

```
public abstract void setLocale(Locale locale);
```

Parameters

`locale`	The locale to set.

B.2.2.2. KlarosContextParameter

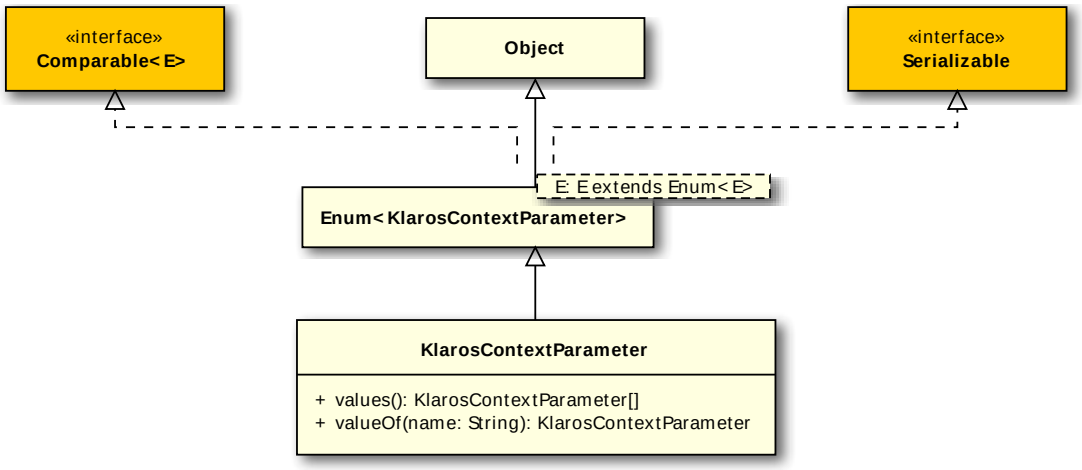
The available context parameters.

B.2.2.2.1. Synopsis

```
public final class KlarosContextParameter extends Enum<KlarosContextParameter> {

  // Public Static Fields
  public static final KlarosContextParameter applicationBaseUrl;
  public static final KlarosContextParameter date;
  public static final KlarosContextParameter imageProvider;
  public static final KlarosContextParameter parameters;
  public static final KlarosContextParameter user;

  // Public Static Methods
  public static KlarosContextParameter valueOf(String name);
  public static KlarosContextParameter[] values();
}
```

Methods inherited from java.lang.Enum: `compareTo` , `equals` , `getDeclaringClass` , `hashCode` , `name` , `ordinal` , `toString` , `valueOf`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `wait`



B.2.2.2.2. applicationBaseUrl

```
public static final KlarosContextParameter applicationBaseUrl;
```

The application base url.

B.2.2.2.3. date

```
public static final KlarosContextParameter date;
```

The current date.

B.2.2.2.4. imageProvider

```
public static final KlarosContextParameter imageProvider;
```

The image provider.

B.2.2.2.5. parameters

```
public static final KlarosContextParameter parameters;
```

The parameters.

B.2.2.2.6. user

```
public static final KlarosContextParameter user;
```

The current user.

B.2.3. de.verit.klaros.scripting.custom

B.2.3.1. IterationsReport

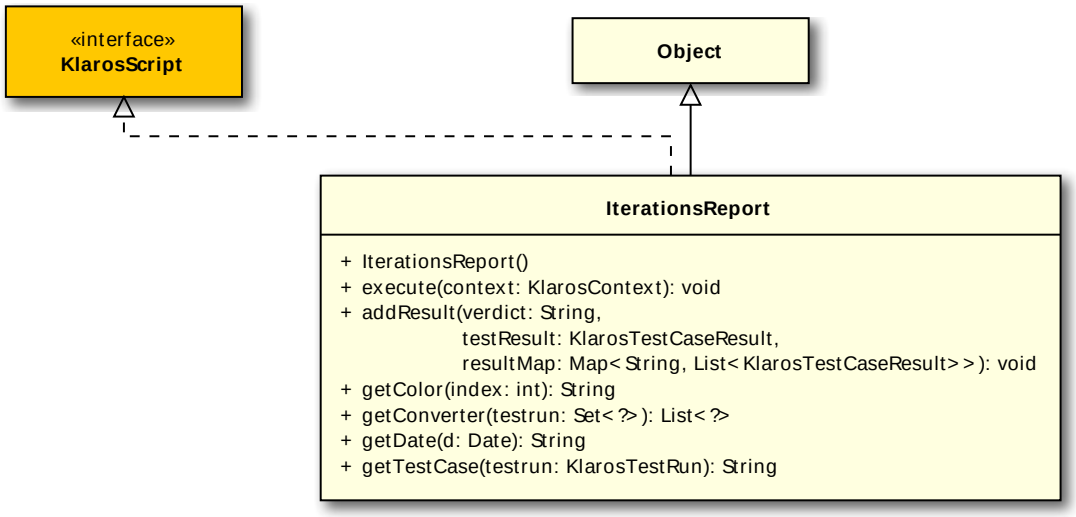
The Class IterationsReport. A Java snippet to create an Iteration Report Template.

B.2.3.1.1. Synopsis

```
public class IterationsReport implements KlarosScript {
```

```
  // Public Constructors
  public IterationsReport();

  // Public Methods
  public void addResult(String verdict,
                  KlarosTestCaseResult testResult,
                  Map<String, List<KlarosTestCaseResult>> resultMap);
  public void execute(KlarosContext context);
  public String getColor(int index);
  public List<?> getConverter(Set<?> testrun);
  public String getDate(Date d);
  public String getTestCase(KlarosTestRun testrun);
}
```

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.2.3.1.2. addResult(String, KlarosTestCaseResult, Map<String, List<KlarosTestCaseResult>>)

```
public void addResult(String verdict,
                  KlarosTestCaseResult testResult,
                  Map<String, List<KlarosTestCaseResult>> resultMap);
```

Adds the result to the results map.

Parameters

`verdict`	the verdict of the result
`testResult`	the test result
`resultMap`	the result map

B.2.3.1.3. getColor(int)

```
public String getColor(int index);
```

Gets the color.

Parameters

`index`	the index
`return`	the color

B.2.3.1.4. getConverter(Set<?>)

```
public List<?> getConverter(Set<?> testrun);
```

Gets the converter.

Parameters

`testrun`	the Test Run
`return`	the converter

B.2.3.1.5. getDate(Date)

```
public String getDate(Date d);
```

Gets the date.

Parameters

`d`	the date
`return`	the date

B.2.3.1.6. getTestCase(KlarosTestRun)

```
public String getTestCase(KlarosTestRun testrun);
```

Gets the test case.

Parameters

`testrun`	the Test Run
`return`	the Test Case Name

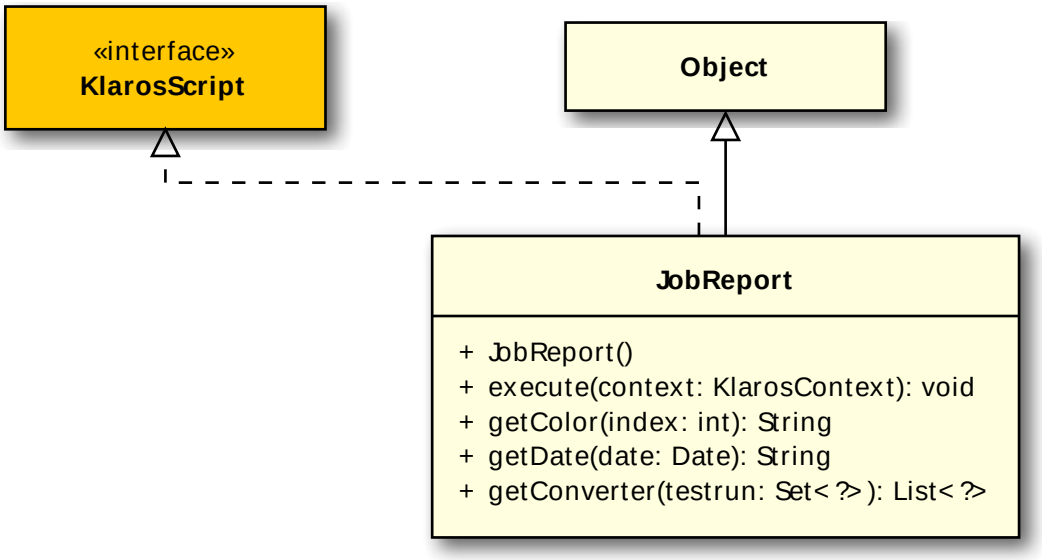
B.2.3.2. JobReport

The Class JobReport. A Java snippet to create an Job Report Template.

B.2.3.2.1. Synopsis

```
public class JobReport implements KlarosScript {

  // Public Constructors
  public JobReport();

  // Public Methods
  public void execute(KlarosContext context);
  public String getColor(int index);
  public final List<?> getConverter(Set<?> testrun);
  public String getDate(Date date);
}
```

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.2.3.2.2. getColor(int)

```
public String getColor(int index);
```

Return the background color code for the given table row index. This method implements a zebra color schema with alternating colors for odd and even rows.

Parameters

`index`	the row index
`return`	the HTML color code string

B.2.3.2.3. getConverter(Set<?>)

```
public final List<?> getConverter(Set<?> testrun);
```

Converts Collections into a ArrayList.

Parameters

`testrun`	the testrun
`return`	the converter

B.2.3.2.4. getDate(Date)

```
public String getDate(Date date);
```

Return the formatted date string of the given date.

Parameters

`date`	the date
`return`	the formatted date string

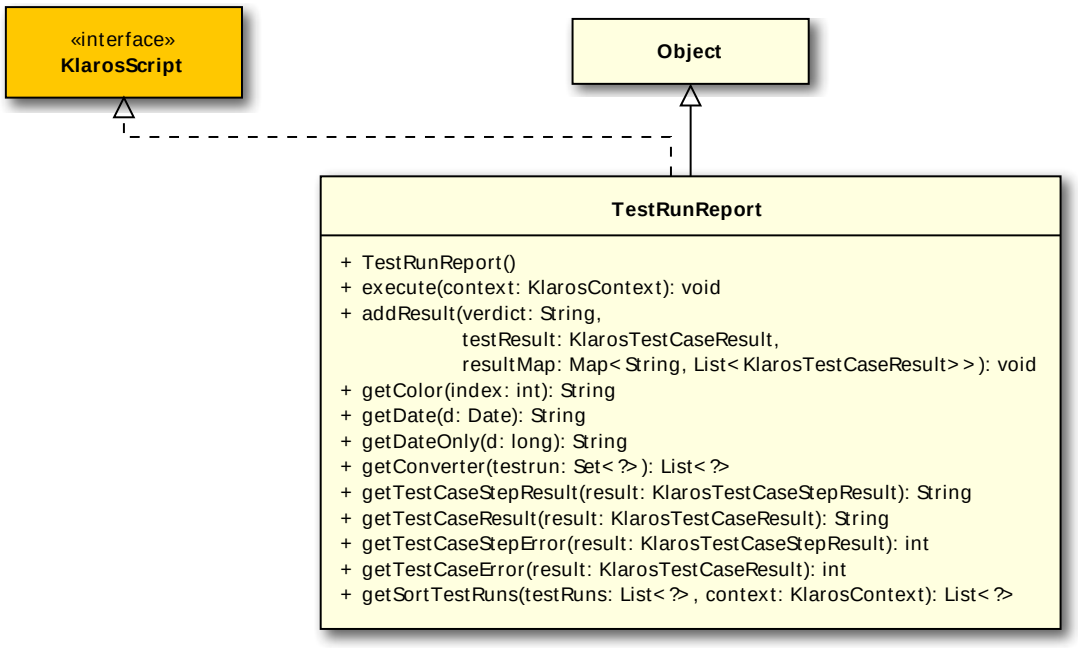
B.2.3.3. TestRunReport

The Class TestRunReport. A Java snippet to create an Test Run Report Template.

B.2.3.3.1. Synopsis

```
public class TestRunReport implements KlarosScript {

  // Public Constructors
  public TestRunReport();

  // Public Methods
  public void addResult(String verdict,
                  KlarosTestCaseResult testResult,
                  Map<String, List<KlarosTestCaseResult>> resultMap);
  public final void execute(KlarosContext context);
  public final String getColor(int index);
  public final List<?> getConverter(Set<?> testrun);
  public final String getDate(Date d);
  public final String getDateOnly(long d);
  public final List<?> getSortTestRuns(List<?> testRuns, KlarosContext context);
  public final int getTestCaseError(KlarosTestCaseResult result);
  public final String getTestCaseResult(KlarosTestCaseResult result);
  public final int getTestCaseStepError(KlarosTestCaseStepResult result);
  public final String getTestCaseStepResult(KlarosTestCaseStepResult result);
}
```

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.2.3.3.2. addResult(String, KlarosTestCaseResult, Map<String, List<KlarosTestCaseResult>>)

```
public void addResult(String verdict,
```

```
                    KlarosTestCaseResult testResult,
                    Map<String, List<KlarosTestCaseResult>> resultMap);
```

Adds the result to the results map.

Parameters

`verdict`	the verdict of the result
`testResult`	the test result
`resultMap`	the result map

B.2.3.3.3. getColor(int)

```
public final String getColor(int index);
```

Gets the color.

Parameters

`index`	the index
`return`	the color

B.2.3.3.4. getConverter(Set<?>)

```
public final List<?> getConverter(Set<?> testrun);
```

Gets the converter.

Parameters

`testrun`	the Test Run
`return`	the converter

B.2.3.3.5. getDate(Date)

```
public final String getDate(Date d);
```

Gets the date.

Parameters

`d`	the d
`return`	the date

B.2.3.3.6. getDateOnly(long)

```
public final String getDateOnly(long d);
```

Gets the date only.

Parameters

`d`	the d
`return`	the date only

B.2.3.3.7. getSortTestRuns(List<?>, KlarosContext)

```
public final List<?> getSortTestRuns(List<?> testRuns, KlarosContext context);
```

Gets the sort test runs.

Parameters

`testRuns`	the test runs
`context`	the context
`return`	the sort test runs

B.2.3.3.8. getTestCaseError(KlarosTestCaseResult)

```
public final int getTestCaseError(KlarosTestCaseResult result);
```

Gets the error.

Parameters

`result`	the result
`return`	the error

B.2.3.3.9. getTestCaseResult(KlarosTestCaseResult)

```
public final String getTestCaseResult(KlarosTestCaseResult result);
```

Gets the result.

Parameters

`result`	the result
`return`	the result

B.2.3.3.10. getTestCaseStepError(KlarosTestCaseStepResult)

```
public final int getTestCaseStepError(KlarosTestCaseStepResult result);
```

Gets the error.

Parameters

`result`	the result
`return`	the error

B.2.3.3.11. getTestCaseStepResult(KlarosTestCaseStepResult)

```
public final String getTestCaseStepResult(KlarosTestCaseStepResult result);
```

Gets the result.

Parameters

`result`	the result
`return`	the result

B.2.4. de.verit.klaros.scripting.model

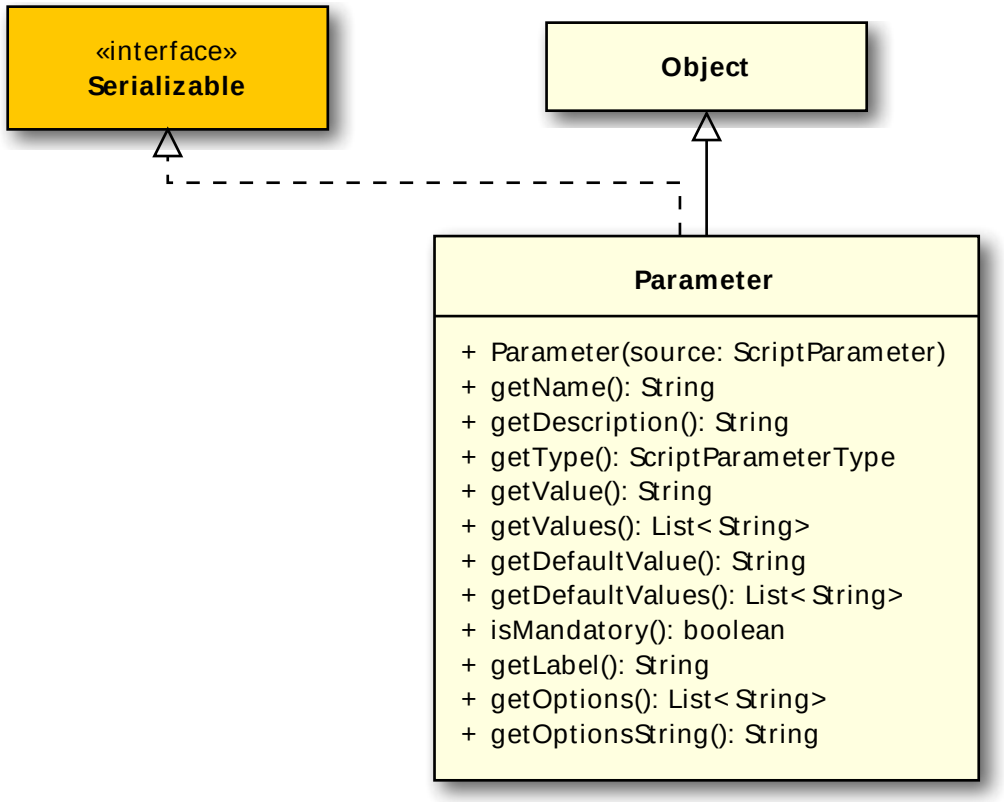
B.2.4.1. Parameter

The Parameter class encapsulates a report query parameter.

B.2.4.1.1. Synopsis

```
public class Parameter implements Serializable {

  // Public Constructors
  public Parameter(ScriptParameter source);

  // Public Methods
  public String getDefaultValue();
  public List<String> getDefaultValues();
  public String getDescription();
  public String getLabel();
  public String getName();
  public List<String> getOptions();
  public String getOptionsString();
  public ScriptParameterType getType();
  public String getValue();
  public List<String> getValues();
  public boolean isMandatory();
}
```

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.2.4.1.2. Parameter(ScriptParameter)

```
public Parameter(ScriptParameter source);
```

Instantiates a new parameter.

Parameters

`source`	the source script parameter

B.2.4.1.3. getDefaultValue()

```
public String getDefaultValue();
```

Gets the default value.

Parameters

`return`	the default value

B.2.4.1.4. getDefaultValues()

```
public List<String> getDefaultValues();
```

Gets the default values.

Parameters

`return`	the default values

B.2.4.1.5. getDescription()

```
public String getDescription();
```

Gets the description.

Parameters

`return`	the description

B.2.4.1.6. getLabel()

```
public String getLabel();
```

Gets the label.

Parameters

`return`	the label

B.2.4.1.7. getName()

```
public String getName();
```

Gets the name.

Parameters

`return`	the name

B.2.4.1.8. getOptions()

```
public List<String> getOptions();
```

Gets the options.

Parameters

`return` the options

B.2.4.1.9. getOptionsString()

```
public String getOptionsString();
```

Gets the options string.

Parameters

`return` the options string

B.2.4.1.10. getType()

```
public ScriptParameterType getType();
```

Gets the type.

Parameters

`return` the type

B.2.4.1.11. getValue()

```
public String getValue();
```

Gets the value.

Parameters

`return` the value

B.2.4.1.12. getValues()

```
public List<String> getValues();
```

Gets the value.

Parameters

`return` the value

B.2.4.1.13. isMandatory()

```
public boolean isMandatory();
```

Checks if is mandatory.

Parameters

`return` true, if is mandatory

B.2.4.2. ParameterContext

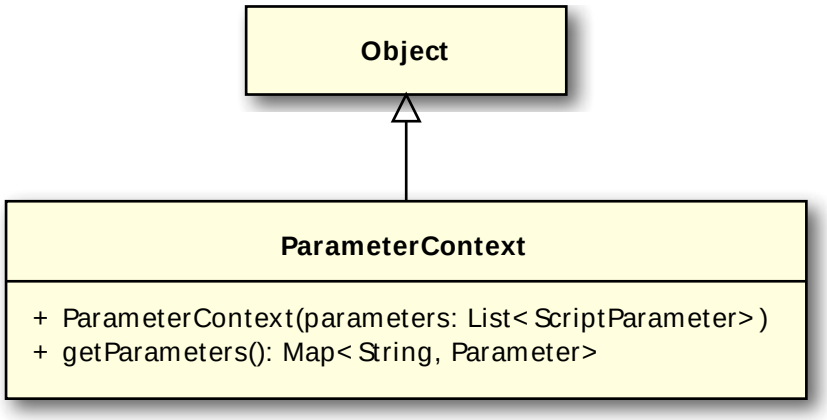
The context for script parameters.

B.2.4.2.1. Synopsis

```
public class ParameterContext {
```

```
  // Public Constructors
  public ParameterContext(List<ScriptParameter> parameters);

  // Public Methods
  public Map<String, Parameter> getParameters();
}
```

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.2.4.2.2. ParameterContext(List<ScriptParameter>)

```
public ParameterContext(List<ScriptParameter> parameters);
```

Create a ParameterContext.

Parameters

`parameters`	the jcr script parameters

B.2.4.3. ParameterType

The possible script parameter types.

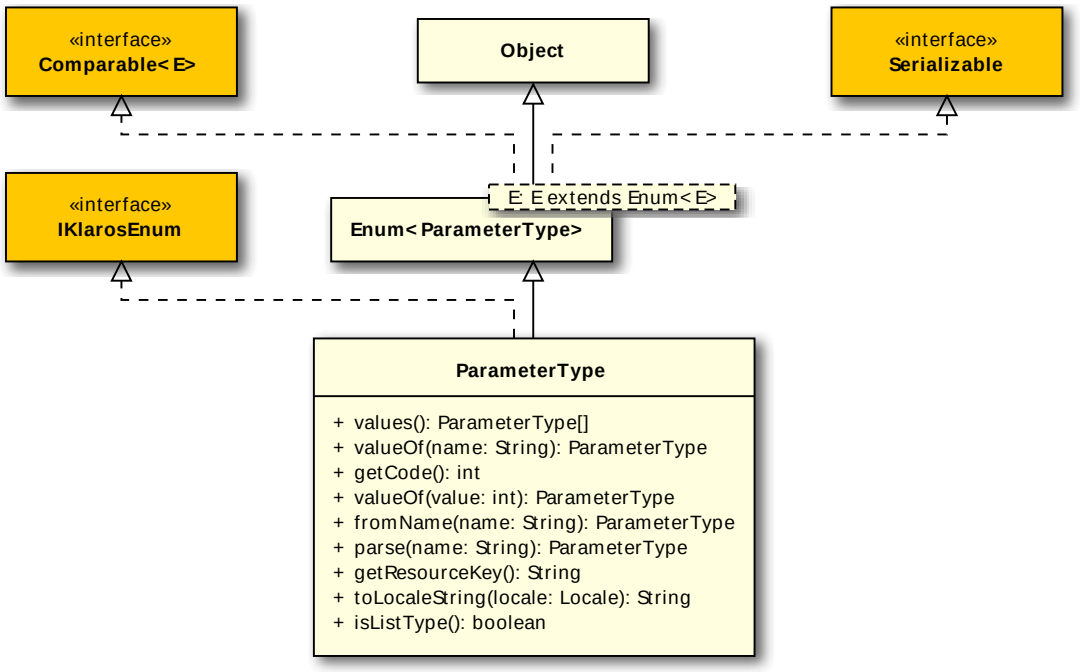
B.2.4.3.1. Synopsis

```
public final class ParameterType extends Enum<ParameterType>
   implements IKlarosEnum {

  // Public Static Fields
  public static final ParameterType BOOLEAN;
  public static final ParameterType DATE;
  public static final ParameterType INTEGER;
  public static final ParameterType LIST;
  public static final ParameterType MULTI_LIST;
  public static final ParameterType STRING;

  // Public Static Methods
  public static ParameterType fromName(String name);
```

```
  public static ParameterType parse(String name);
  public static ParameterType valueOf(int value);
  public static ParameterType valueOf(String name);
  public static ParameterType[] values();

  // Public Methods
  public int getCode();
  public String getResourceKey();
  public boolean isListType();
  public String toLocaleString(Locale locale);
}
```

Methods inherited from java.lang.Enum: `compareTo` , `equals` , `getDeclaringClass` , `hashCode` , `name` , `ordinal` , `toString` , `valueOf`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `wait`



B.2.4.3.2. fromName(String)

```
  public static ParameterType fromName(String name);
```

Create enum from a name string.

Parameters

`name`	the name
`return`	An enum by its name.

B.2.4.3.3. getResourceKey()

```
  public String getResourceKey();
```

Gets the resource key.

Parameters

`return`	the resource key

B.2.4.3.4. isListType()

```
public boolean isListType();
```

Checks if this type is a list type.

Parameters

`return`	true, if list type

B.2.4.3.5. parse(String)

```
public static ParameterType parse(String name);
```

Parse an instance from a name string.

Parameters

`name`	the name of the enumeration as returned from toString() or name() methods.
`return`	The enumeration or null if no match.

B.2.4.3.6. valueOf(int)

```
public static ParameterType valueOf(int value);
```

Create from a code value.

Parameters

`value`	the code value
`return`	Returns the value of the given code.

B.2.5. Constant field values

B.2.5.1. de.verit.klaros.scripting.*

WRAPPER_PREFIX	"Klaros"

Table B.1. KlarosQueryFactory

Appendix C. Test Case Import File Specification

The following table gives a short overview of the available elements:

Name	Description
<areatopic>	The category of requirements that is covered by the test case.
<attribute>	A custom attribute.
<attributes>	The list of custom attributes.
<container>	The container element for test cases.
<depends>	The string describing the dependencies of this test case.
<description>	The textual description of the test case.
<docbase>	A reference to the document this test case is based on, which may e.
<evaluation>	The evaluation method of this test case.
<execution>	The execution type of this test case, manual or automated.
<expectedResult>	The expected result of this test.
<externalId>	The external id of this test case.
<externalRevision>	The external revision of this test case.
<level>	The level of this test case.
<method>	The test design view on the system under test.
<name>	The name of the custom attribute.
<note>	A free text short note for this test case.
<postcondition>	The postcondition for this test case.
<precondition>	The precondition for this test case.
<priority>	The priority of this test case.
<revision>	The revision of this test case.
<shortname>	The editable title of the test case.
<state>	The state of this test case.
<step>	A test case step.
<steps>	The list of contained test case steps.
<team>	The team this test case is assigned to.
<testcase>	A test case definition.
<testcases>	The list of custom attributes.
<testsuite>	If this element is present, a test suite with the name given here will be generated upon import.
<traceability>	The traceability information for this test case.
<trigger>	The externally initiated trigger for this test case.

Name	Description
<triggers>	The list of triggers for this object.
<value>	The value of the custom attribute.
<variety>	The result expectation of the test: Whether a positive or negative result is expected.

Table C.1. Element summary

C.1. <step>

A test case step.

This element may occur inside the following elements: <steps> .

Name	Multiplicity	Description
<description>	0..1	
<expectedResult>	0..1	
<postcondition>	0..1	
<precondition>	0..1	

Table C.2. <step> elements

C.2. <steps>

The list of contained test case steps.

This element may occur inside the following elements: <testcase> .

Name	Multiplicity	Description
<step>	0..n	

Table C.3. <steps> elements

C.3. <attribute>

A custom attribute.

This element may occur inside the following elements: <attributes> .

Name	Multiplicity	Description
<name>	1..1	
<value>	1..1	

Table C.4. <attribute> elements

C.4. <attributes>

The list of custom attributes.

This element may occur inside the following elements: <testcase> .

Name	Multiplicity	Description
<attribute>	0..n	

Table C.5. <attributes> elements

C.5. <container>

The container element for test cases. This is the root element of this schema.

Name	Multiplicity	Description
<testsuite>	0..1	
<testcases>	0..1	

Table C.6. <container> elements

C.6. <testcases>

The list of custom attributes.

This element may occur inside the following elements: <container> .

Name	Multiplicity	Description
<testcase>	0..n	

Table C.7. <testcases> elements

C.7. <testcase>

A test case definition.

This element may occur inside the following elements: <testcases> .

Name	Multiplicity	Description
<areatopic>	0..1	
<attributes>	0..1	
<depends>	0..1	
<description>	0..1	
<docbase>	0..1	
<evaluation>	0..1	
<execution>	0..1	
<expectedResult>	0..1	
<externalId>	0..1	
<externalRevision>	0..1	
<level>	0..1	
<method>	0..1	
<note>	0..1	
<postcondition>	0..1	
<precondition>	0..1	
<priority>	0..1	

Name	Multiplicity	Description
<revision>	0..1	
<shortname>	0..1	
<state>	0..1	
<steps>	0..1	
<team>	0..1	
<traceability>	0..1	
<trigger>	0..1	
<variety>	0..1	

Table C.8. <testcase> elements

C.8. <triggers>

The list of triggers for this object.

This element may occur inside the following elements: <testcase> .

Name	Multiplicity	Description
<trigger>	0..n	

Table C.9. <triggers> elements

C.9. <testsuite>

If this element is present, a test suite with the name given here will be generated upon import. This test suite will contain all the test cases that are part of this import container.

This element may occur inside the following elements: <container> .

C.10. <externalId>

The external id of this test case. Reserved for future use.

This element may occur inside the following elements: <testcase> .

C.11. <externalRevision>

The external revision of this test case. Reserved for future use.

This element may occur inside the following elements: <testcase> .

C.12. <trigger>

The externally initiated trigger for this test case. Reserved for future use.

This element may occur inside the following elements: <triggers> .

C.13. <areatopic>

The category of requirements that is covered by the test case. FUNCTIONAL(100, "tc_functional"), // NON_FUNCTIONAL(200, "tc_non_functional"), // STRUCTURAL(300, "tc_structure"), // REGRESSION(400, "tc_regression"), // RE_TEST

This element may occur inside the following elements: <testcase> .

C.14. <depends>

The string describing the dependencies of this test case.

This element may occur inside the following elements: <testcase> .

C.15. <description>

The textual description of the test case.

This element may occur inside the following elements: <step> , <testcase> .

C.16. <docbase>

A reference to the document this test case is based on, which may e.g. contain the requirements this test case is related to.

This element may occur inside the following elements: <testcase> .

C.17. <evaluation>

The evaluation method of this test case.

This element may occur inside the following elements: <testcase> .

C.18. <execution>

The execution type of this test case, manual or automated.

This element may occur inside the following elements: <testcase> .

C.19. <expectedResult>

The expected result of this test.

This element may occur inside the following elements: <step> , <testcase> .

C.20. <level>

The level of this test case.

This element may occur inside the following elements: <testcase> .

C.21. <method>

The test design view on the system under test.

This element may occur inside the following elements: <testcase> .

C.22. <note>

A free text short note for this test case.

This element may occur inside the following elements: <testcase> .

C.23. <postcondition>

The postcondition for this test case.

This element may occur inside the following elements: <step> , <testcase> .

C.24. <precondition>

The precondition for this test case.

This element may occur inside the following elements: <step> , <testcase> .

C.25. <revision>

The revision of this test case. This field is used only for exporting test cases, imported test cases always have the fixed revision ID "1.0".

This element may occur inside the following elements: <testcase> .

C.26. <priority>

The priority of this test case.

This element may occur inside the following elements: <testcase> .

C.27. <shortname>

The editable title of the test case.

This element may occur inside the following elements: <testcase> .

C.28. <state>

The state of this test case. This information is ignored for import operations.

This element may occur inside the following elements: <testcase> .

C.29. <team>

The team this test case is assigned to.

This element may occur inside the following elements: <testcase> .

C.30. <traceability>

The traceability information for this test case.

This element may occur inside the following elements: <testcase> .

C.31. <variety>

The result expectation of the test: Whether a positive or negative result is expected.

This element may occur inside the following elements: <testcase> .

C.32. <name>

The name of the custom attribute.

This element may occur inside the following elements: <attribute> .

C.33. <value>

The value of the custom attribute.

This element may occur inside the following elements: <attribute> .

Appendix D. Requirement Import File Specification

The following table gives a short overview of the available elements:

Name	Description
<attribute>	A custom attribute.
<attributes>	The list of custom attributes.
<container>	The container element for requirements.
<description>	The description of this requirement.
<externalId>	The external id of this requirement, the format of this string can be chosen deliberately.
<externalRevision>	The external revision of this requirement, the format of this string can be chosen deliberately.
<name>	The name of the custom attribute.
<priority>	The priority of this requirement.
<requirement>	A requirement definition.
<requirements>	The list of custom attributes.
<revision>	The revision of this requirement.
<shortname>	The short name / code of this object.
<summary>	The summary of this requirement.
<trigger>	The externally initiated trigger for this requirement.
<triggers>	The list of triggers for this object.
<value>	The value of the custom attribute.

Table D.1. Element summary

D.1. <attribute>

A custom attribute.

This element may occur inside the following elements: <attributes> .

Name	Multiplicity	Description
<name>	1..1	
<value>	1..1	

Table D.2. <attribute> elements

D.2. <attributes>

The list of custom attributes.

Name	Multiplicity	Description
<attribute>	0..n	

Table D.3. <attributes> elements

D.3. <container>

The container element for requirements. This is the root element of this schema.

Name	Multiplicity	Description
<requirements>	0..1	

Table D.4. <container> elements

D.4. <requirements>

The list of custom attributes.

This element may occur inside the following elements: <container> .

Name	Multiplicity	Description
<requirement>	0..n	

Table D.5. <requirements> elements

D.5. <requirement>

A requirement definition.

This element may occur inside the following elements: <requirements> .

Name	Multiplicity	Description
<description>	0..1	

Name	Multiplicity	Description
<externalId>	0..1	
<externalRevision>	0..1	
<priority>	0..1	
<revision>	0..1	
<shortname>	0..1	
<summary>	0..1	
<triggers>	0..1	

Table D.6. <requirement> elements

D.6. <triggers>

The list of triggers for this object.

This element may occur inside the following elements: <requirement> .

Name	Multiplicity	Description
<trigger>	0..n	

Table D.7. <triggers> elements

D.7. <externalId>

The external id of this requirement, the format of this string can be chosen deliberately. This element is ignored for import operations and will only be used during synchronization. Changing this id for subsequent synchronization attempts will result in the creation of additional requirements. Else the data in the requirement will be overwritten with the data provided.

This element may occur inside the following elements: <requirement> .

D.8. <externalRevision>

The external revision of this requirement, the format of this string can be chosen deliberately. This element is ignored for import operations and will only be used during synchronization. Changing this revision for subsequent synchronization attempts may result in the creation of an additional revision of the requirement if the revision is yet unknown. Else the data in the requirement will be overwritten with the data provided.

This element may occur inside the following elements: <requirement> .

D.9. <trigger>

The externally initiated trigger for this requirement. Reserved for future use.

This element may occur inside the following elements: <triggers> .

D.10. <description>

The description of this requirement.

This element may occur inside the following elements: <requirement> .

D.11. <priority>

The priority of this requirement.

This element may occur inside the following elements: <requirement> .

D.12. <revision>

The revision of this requirement. This field is used only for exporting requirements, imported requirements always have the revision ID "1.0".

This element may occur inside the following elements: <requirement> .

D.13. <shortname>

The short name / code of this object.

This element may occur inside the following elements: <requirement> .

D.14. <summary>

The summary of this requirement.

This element may occur inside the following elements: <requirement> .

D.15. <name>

The name of the custom attribute.

This element may occur inside the following elements: <attribute> .

D.16. <value>

The value of the custom attribute.

This element may occur inside the following elements: <attribute> .

Appendix E. Reporting Resources

E.1. Context Variables

Name	Description
date	The current date.
locale	The current locale set in the web frontend.
activeProject	The currently selected `KlarosConfiguration`. If no project is selected this variable contains the null object.
user	A `KlarosUser` object representing the active user.
parameters	A Map of parameter objects representing the entered parameters when running the report. The map key consists of the parameter name attribute the value is of type `de.verit.klaros.scripting.model.Parameter`.

Table E.1. Context Variables

The context variables can be accessed via SeamPDF by e.g.:

```
<p:text value="#{date}" />
```

Note

#{user.name} and #{user.username} contain different values. The first provides the user's real name, while the latter provides the name the user is logged in with.

E.2. KlarosScript Interface

```
package de.verit.klaros.scripting;
/**
 * Public interface that all seam-pdf template scripts must implement be work
 * properly.
 */
public interface KlarosScript {
    /**
     * This functions gets called by the seam-pdf servlet to execute the script.
     *
     * @param context
     *            The event context to provide all needed functions, properties
     *            and objects.
     */
    void execute(KlarosContext context);
}
```

E.3. Example report template

```
<p:document xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:f="http://java.sun.com/jsf/core" xmlns:p="http://jboss.org/schema/seam/pdf"
  title="Klaros-Testmanagement Test Suite Report" marginMirroring="true"
  author="#{user.name}" creator="#{user.name}" pageSize="A4">
```

```
<f:facet name="header">
  <p:font size="8">
    <p:header borderWidthBottom="0.1" borderColorBottom="black" borderWidthTop="0" alignment="center">
      <p:text value="Example report - generated #{date} by #{user.name}"/>
    </p:header>
    <p:footer borderWidthTop="0.1" borderColorTop="black" borderWidthBottom="0" alignment="center">
      <p:text value="Page " />
      <p:pageNumber />
    </p:footer>
  </p:font>
</f:facet>
<!-- print the frontpage -->
<p:paragraph alignment="center" spacingAfter="100">
  <p:text value="" />
</p:paragraph>
<p:font style="bold" size="32">
  <p:paragraph alignment="center" spacingAfter="75">
    <p:text value="Test Case Report" />
  </p:paragraph>
</p:font>
<p:font style="normal" size="12">
  <p:paragraph alignment="center" spacingAfter="5">
    <p:text value="Created by" />
  </p:paragraph>
</p:font>
<p:font style="bold" size="16">
  <p:paragraph alignment="center" spacingAfter="5">
    <p:text value="#{user.name} (#{user.email})"/>
  </p:paragraph>
</p:font>
<p:font style="normal" size="12">
  <p:paragraph alignment="center" spacingAfter="5">
    <p:text value="at" />
  </p:paragraph>
</p:font>
<p:font style="bold" size="16">
  <p:paragraph alignment="center" spacingAfter="75">
    <p:text value="#{date}" />
  </p:paragraph>
</p:font>
<p:newPage/>
<ui:fragment rendered="#{results != null}">
  <p:font style="normal" size="12">
    <p:paragraph alignment="left" spacingAfter="10">
      <p:text value="The testresults for " />
    </p:paragraph>
  </p:font>
</ui:fragment>
<!-- Testresult table -->
<p:table columns="4" widths="1 1 3 3">
  <!-- create the headline with bold characters -->
  <p:font size="10" style="bold">
    <p:cell horizontalAlignment="center" verticalAlignment="top">
      <p:paragraph>
        <p:text value="Name" />
      </p:paragraph>
    </p:cell>
    <p:cell horizontalAlignment="center" verticalAlignment="top">
      <p:paragraph>
        <p:text value="Result" />
      </p:paragraph>
```

```
    </p:cell>
    <p:cell horizontalAlignment="center" verticalAlignment="top">
      <p:paragraph>
        <p:text value="Testrun description" />
      </p:paragraph>
    </p:cell>
    <p:cell horizontalAlignment="center" verticalAlignment="top">
      <p:paragraph>
        <p:text value="Summary" />
      </p:paragraph>
    </p:cell>
  </p:font>
  <!-- display the attributes of the test results -->
  <p:font size="8">
    <ui:repeat value="#{results}" var="tcr">
      <p:cell verticalAlignment="top" horizontalAlignment="left">
        <p:paragraph>
          <p:text value="#{tcr.testCase.name}" />
        </p:paragraph>
      </p:cell>
      <!-- decide which color has to be displayed, based on the testresult -->
      <ui:fragment rendered="#{tcr.error}">
        <p:cell backgroundColor="rgb(255,0,0)" verticalAlignment="top" horizontalAlignment="center">
          <p:paragraph>
            <p:text value="error" />
          </p:paragraph>
        </p:cell>
      </ui:fragment>
      <ui:fragment rendered="#{tcr.failure}">
        <p:cell backgroundColor="rgb(255,215,0)" verticalAlignment="top" horizontalAlignment="center">
          <p:paragraph>
            <p:text value="failure" />
          </p:paragraph>
        </p:cell>
      </ui:fragment>
      <ui:fragment rendered="#{tcr.passed}">
        <p:cell backgroundColor="rgb(0,255,0)" verticalAlignment="top" horizontalAlignment="center">
          <p:paragraph>
            <p:text value="passed" />
          </p:paragraph>
        </p:cell>
      </ui:fragment>
      <p:cell verticalAlignment="top" horizontalAlignment="left">
        <p:paragraph>
          <p:text value="#{tcr.description}" />
        </p:paragraph>
      </p:cell>
      <p:cell verticalAlignment="top" horizontalAlignment="left">
        <p:paragraph>
          <p:text value="#{tcr.summary}" />
        </p:paragraph>
      </p:cell>
      <!-- Print the testcase description below the result row.
           To differ from the next row use a bigger border for the bottom. -->
      <p:cell colspan="4" verticalAlignment="top" horizontalAlignment="left"
              borderWidthBottom="1" paddingBottom="3">
        <p:paragraph>
          <p:font size="6" style="bold">
            <p:text value="Testcase description:" />
          </p:font>
          <p:font size="6">
```

```
            <p:text value="#{tcr.testCase.description}" />
          </p:font>
        </p:paragraph>
      </p:cell>
    </ui:repeat>
  </p:font>
 </p:table>
</p:document>
```

Index

A

applicationBaseUrl, 330

B

Back, 117

C

Categories, 57
Classes
- IterationsReport, 330
- JobReport, 332
- KlarosAttachment, 277
- KlarosCategoryNode, 278
- KlarosCategoryTree, 279
- KlarosConfiguration, 280
- KlarosContext, 322
- KlarosContextParameter, 329
- KlarosEnumValue, 282
- KlarosExternalImplementation, 282
- KlarosIssue, 283
- KlarosIteration, 284
- KlarosIterationCategoryNode, 285
- KlarosIterationCategoryTree, 286
- KlarosJob, 287
- KlarosJobTimeBlock, 289
- KlarosJobUpdateAction, 290
- KlarosLabeledObject, 291
- KlarosNamedEntity, 292
- KlarosProperty, 293
- KlarosQueryFactory, 324
- KlarosRequirement, 294
- KlarosRequirementCategoryNode, 295
- KlarosRequirementCategoryTree, 296
- KlarosRequirementGroup, 297
- KlarosRevision, 298
- KlarosSUTImplementation, 299
- KlarosSUTImplementationCategoryNode, 301
- KlarosSUTImplementationCategoryTree, 301
- KlarosTag, 302
- KlarosTestCase, 303
- KlarosTestCaseCategoryNode, 305
- KlarosTestCaseCategoryTree, 306
- KlarosTestCaseResult, 307
- KlarosTestCaseState, 308
- KlarosTestCaseStep, 309
- KlarosTestCaseStepResult, 310
- KlarosTestEnvironment, 311
- KlarosTestEnvironmentCategoryNode, 312
- KlarosTestEnvironmentCategoryTree, 313
- KlarosTestExecutable, 314
- KlarosTestRun, 315
- KlarosTestSuite, 316
- KlarosTestSuiteCategoryNode, 318
- KlarosTestSuiteCategoryTree, 319
- KlarosTestSuiteResult, 320
- KlarosUser, 321
- Parameter, 336
- ParameterContext, 339
- ParameterType, 340
- TestRunReport, 334

Create Issue, 117, 118

D

date, 330

E

Error, 116, 118
Expected Result, 97

F

Failure, 116, 118
Fields
- applicationBaseUrl, 330
- date, 330
- imageProvider, 330
- parameters, 330
- user, 330
- WRAPPER_PREFIX, 325

Finish Run, 118

I

IKlarosAttachment, 216
IKlarosCategoryNode, 217
IKlarosCategoryTree, 219
IKlarosConfiguration, 220
IKlarosContext, 326
IKlarosEnumValue, 224
IKlarosExternalImplementation, 224
IKlarosExternalLink, 225
IKlarosIssue, 226
IKlarosIteration, 228
IKlarosJob, 231
IKlarosJobTimeBlock, 233

IKlarosJobUpdateAction, 235
IKlarosLabeledObject, 236
IKlarosNamedEntity, 237
IKlarosProperty, 238
IKlarosRequirement, 239
IKlarosRequirementGroup, 242
IKlarosRevision, 245
IKlarosSUTImplementation, 247
IKlarosTag, 250
IKlarosTestCase, 251
IKlarosTestCaseResult, 257
IKlarosTestCaseState, 261
IKlarosTestCaseStep, 262
IKlarosTestCaseStepResult, 263
IKlarosTestEnvironment, 267
IKlarosTestExecutable, 268
IKlarosTestRun, 268
IKlarosTestSuite, 272
IKlarosTestSuiteResult, 275
IKlarosUser, 276
imageProvider, 330
Interfaces
  IKlarosAttachment, 216
  IKlarosCategoryNode, 217
  IKlarosCategoryTree, 219
  IKlarosConfiguration, 220
  IKlarosContext, 326
  IKlarosEnumValue, 224
  IKlarosExternalImplementation, 224
  IKlarosExternalLink, 225
  IKlarosIssue, 226
  IKlarosIteration, 228
  IKlarosJob, 231
  IKlarosJobTimeBlock, 233
  IKlarosJobUpdateAction, 235
  IKlarosLabeledObject, 236
  IKlarosNamedEntity, 237
  IKlarosProperty, 238
  IKlarosRequirement, 239
  IKlarosRequirementGroup, 242
  IKlarosRevision, 245
  IKlarosSUTImplementation, 248
  IKlarosTag, 250
  IKlarosTestCase, 251
  IKlarosTestCaseResult, 257
  IKlarosTestCaseState, 261
  IKlarosTestCaseStep, 262
  IKlarosTestCaseStepResult, 263
  IKlarosTestEnvironment, 267
  IKlarosTestExecutable, 268
  IKlarosTestRun, 268
  IKlarosTestSuite, 272
  IKlarosTestSuiteResult, 275
  IKlarosUser, 276
  KlarosScript, 326
Iterations, 73
IterationsReport, 330

J

JobReport, 332

K

KlarosAttachment, 277
KlarosCategoryNode, 278
KlarosCategoryTree, 279
KlarosConfiguration, 280
KlarosContext, 322
KlarosContextParameter, 329
KlarosEnumValue, 281
KlarosExternalImplementation, 282
KlarosIssue, 283
KlarosIteration, 284
KlarosIterationCategoryNode, 285
KlarosIterationCategoryTree, 286
KlarosJob, 287
KlarosJobTimeBlock, 289
KlarosJobUpdateAction, 290
KlarosLabeledObject, 291
KlarosNamedEntity, 292
KlarosProperty, 293
KlarosQueryFactory, 324
KlarosRequirement, 294
KlarosRequirementCategoryNode, 295
KlarosRequirementCategoryTree, 296
KlarosRequirementGroup, 297
KlarosRevision, 298
KlarosScript, 326
KlarosSUTImplementation, 299
KlarosSUTImplementationCategoryNode, 301
KlarosSUTImplementationCategoryTree, 301
KlarosTag, 302
KlarosTestCase, 303
KlarosTestCaseCategoryNode, 305
KlarosTestCaseCategoryTree, 306
KlarosTestCaseResult, 307
KlarosTestCaseState, 308
KlarosTestCaseStep, 309
KlarosTestCaseStepResult, 310

KlarosTestEnvironment, 311
KlarosTestEnvironmentCategoryNode, 312
KlarosTestEnvironmentCategoryTree, 313
KlarosTestExecutable, 314
KlarosTestRun, 315
KlarosTestSuite, 316
KlarosTestSuiteCategoryNode, 318
KlarosTestSuiteCategoryTree, 319
KlarosTestSuiteResult, 320
KlarosUser, 321

L

Link Issue, 117
Login, 49

N

Notification, 171
Notification Scheme, 170

P

Parameter, 336
ParameterContext, 339
parameters, 330
ParameterType, 340
Passed, 116, 118
Postcondition, 95, 97, 116, 118
Precondition, 95, 97, 116, 118
Print Pages, 56
Projects, 68
  Create Project, 69

R

Requirements, 79

S

Skip, 116, 118
Skip All, 116
Systems under Test, 87

T

Test
  Test Case, 114
  Test Suite, 120
Test Cases, 91
Test Environments, 83
Test Execution, 117
Test Suites, 99
TestRunReport, 334

U

user, 330

W

WRAPPER_PREFIX, 325